

Java™ magazine

By and for the Java community 

EMBEDDED EVERYWHERE

THE FUTURE OF CONNECTED
DEVICES STARTS WITH JAVA

31 JAVA ON THE
RASPBERRY PI

26 TOP 10 REASONS
TO USE JAVA IN
EMBEDDED APPS

COMMUNITY

02

From the Editor

04

Java Nation

News, people, and events

14

JCP Executive Series

Q&A with Credit Suisse

EC members on the JCP

JAVA TECH

40

New to Java

Introduction to Web Service Security from Server to Client

The final installment in this series from Max Bonbhel

49

Java Architect

Demystifying invokdynamic

Learn how to use invoke-dynamic in your code.

55

Java Architect

Java Compiler Plug-ins in Java 8

Extend the Java compiler with new behavior.

58

Java Architect

The New javax.cache Caching Standard

The lowdown on javax.cache

62

Enterprise Java

Secure Java EE Authentication

Implementing login authentication using declarative and programmatic security

68

Rich Client

Integrating Web and Java Client Applications with Social Media

Johan Vos gets social.

76

Mobile and Embedded

Getting Started with JSR 281

Bring IP Multimedia Subsystem services to Java-enabled devices.

81

Mobile and Embedded

Swing into Mobile

The Lightweight UI Toolkit and Nokia Series 40 phones

85

Polyglot Programmer

Building Actor-Based Systems Using the Akka Framework

Ted Neward wraps up this two-part series.

92

Fix This

Take our embedded code challenge.



20

EMBEDDED EVERYWHERE

Terrence Barr on Java and the Internet of Things

26

TOP 10 REASONS TO USE JAVA IN EMBEDDED APPS

Why Java is the language of choice

31

Java in Action

JAVA ARRIVES ON A \$25 BOARD

The inspiration behind the Raspberry Pi

36

Java in Action

THE FUTURE OF MONEY

The Royal Canadian Mint banks on Java Card for its digital currency offering.

A woman with blonde hair is sitting at a desk, looking at a computer monitor. She is wearing a dark top. The background shows a large window with blinds, letting in bright light. There are some orange decorative elements on the left side of the frame.

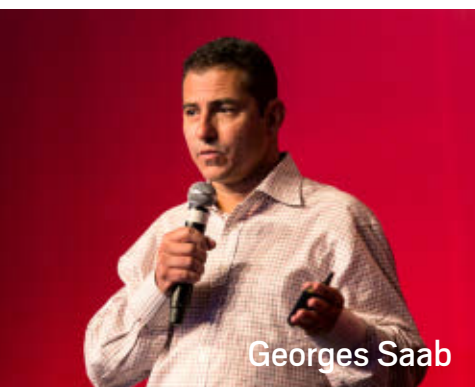
Caroline Kvitka, Editor in Chief BIO



ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// JANUARY/FEBRUARY 2013



Henrik Stahl



Georges Saab

PHOTOGRAPHS BY LUCIANA AITH,
ORQUESTRA DE IMAGENS



The Java band entertains the crowd at the Community keynote.

JAVAONE LOOKS FORWARD

JavaOne Brazil took place in São Paulo, Brazil, December 4–6. Latin America is an important development hub, and it looks as though JavaOne Brazil, now in its third year, is here to stay. “We continually come back to Latin America because of the dedication the community has to driving continued innovation

for Java,” said Oracle’s **Henrik Stahl** in kicking off the Java Strategy and Java Technical keynotes on the first day of the conference. He noted that the success of Java depends on technological innovation, strong stewardship from Oracle, and community participation. “The Latin American Java community (especially in

Brazil) is a shining example of how to be a positive contributor to Java,” he said.

Oracle’s **Georges Saab** discussed some of the recent and upcoming changes to Java. “In addition to the incremental improvements to Java 7, we have also increased the set of platforms supported by Oracle from Linux, Windows, and Oracle Solaris to now also include Mac OS X and Linux/ARM for ARM-based PCs such as the Raspberry Pi and emerging ARM-based microservers.”

Oracle’s **Staffan Friberg** provided an overview of some changes coming in Java 8, including lambda expressions, removal of the permanent generation (PermGen) heap, improved date and time APIs, and improved security.

Oracle’s **Judson Althoff** started his talk off with a bang. “The Internet of Things is on a collision course with big data, and this is a huge opportunity for developers,” he said. Althoff noted that a car embedded with sensors for fuel efficiency, temperature, and tire pressure can generate a petabyte of data a day.

When a “blue screen of death” appeared for the Technical keynote, presenters gladly went on without their slides. What followed was a collection of demos, including JavaFX on a tablet and a look at Project Easel in NetBeans.

Throughout the conference, attendees chose from dozens of sessions, more than half of which were selected by the community, and six hands-on labs.

JAVA IN ACTION



ABOUT US

0

ava
net

olog



05

Geeks Ride Again



Java enthusiasts donned bike helmets and Duke's bike jerseys on the Saturday before JavaOne Brazil for a riding tour of São Paulo. At this now-traditional preconference ride, more than 20 participants enjoyed cycling from Bicycle Park through downtown on street lanes closed off for bicyclists. The ride was 30 kilometers, but participants could opt to take the subway for part of the trip.

PHOTOGRAPH BY TORI WIELDT



JAVA TECH



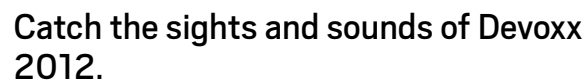
○

ava
net

olog



06



PHOTOGRAPHS COURTESY OF DEVOXX



So far, Devoxx4Kids events are being organized in the Netherlands, the UK, and France.

A photograph of two men standing in front of a large screen. The man on the left is wearing a light blue button-down shirt and khaki pants, holding a small trophy. The man on the right is wearing a dark suit, a light blue shirt, and a yellow tie. The screen behind them shows a close-up of a woman's face, looking upwards. The word "PULSE" is visible on the screen.

A man and a woman are standing next to a dark-colored car. The man, on the right, is wearing a dark jacket and is gesturing with his hands while speaking. The woman, on the left, is wearing a blue t-shirt and glasses. They are in front of a backdrop that features the DevOps conference logo, which includes a stylized coffee cup with a Wi-Fi symbol above it. The text 'DEVOPS conference' is visible on the backdrop. A play button icon is overlaid on the bottom left of the image.

Oracle's Yolande Poirier
chats with Stephen Chin.

JAVA IN ACTION



ABOUT US

0

f

ava
net

log



08

Most survey respondents viewed M2M data pipelines as being critically important. A full **75 percent** said their M2M projects are utilized to create new services. However, **85 percent** noted that the large volumes of data that are passed from the remote embedded devices to the data center create new performance issues. Efficiently filtering and processing the data to enable its use for making timely decisions was cited as another difficulty.

ART BY I-HUA CHEN



Fecak has thought a great deal about the JUG leader role. "I would encourage JUG leaders to always be thinking of how to help their membership. When [vendors] approach me about sponsoring a meeting, I always ask what I can get for my members. Sponsors want to reach audiences, and many sponsors understand that they are building corporate goodwill in the community for their brand." However, he notes that PhillyJUG does not allow pure product demonstrations or any sales activity.

OptionsCity Wins Chicago Innovation Award

Within six years, OptionsCity has gone from unknown to being one of the big players in the options trading platform space in Chicago and New York. Need reliable, time-critical mathematical algorithm performance in your startup product? Think Java.

In the event that you are not ready to migrate to Java SE 7, Oracle offers Java SE support for continued access to critical bug fixes and security fixes as well as general maintenance for JDK 6. Additionally, [Oracle Java SE Advanced](#) and [Oracle Java SE Suite](#) offer superior diagnostics and manageability tools that minimize the costs of deployment, monitoring, and maintenance of Java-based IT environments.



ADAM BIEN

Bien: Lots of interesting things will happen in the near future for Java: JavaFX, Project Nashorn, lambdas, Java EE 7. And I cannot wait for Jigsaw. Java's future is bright.

Our favorite Java authors give you the inside scoop on their books.

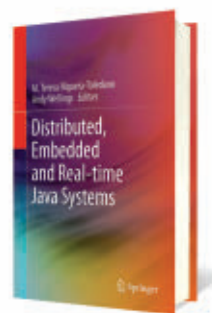


Benjamin Evans
and **Martijn Verburg**
discuss *The Well-
Grounded Java
Developer.*



Jim Weaver and
Stephen Chin discuss
Pro JavaFX 2.

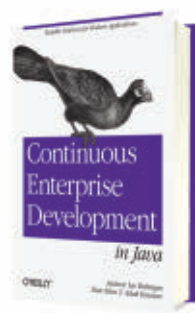
JAVA BOOKS



DISTRIBUTED, EMBEDDED AND REAL-TIME JAVA SYSTEMS

M. Teresa Higuera-Toledano and
Andy J. Wellings (Editors)
Springer (February 2012)

This book is aimed primarily at researchers in real-time embedded systems, particularly those who wish to understand the current state of the art in using Java in this domain. Much of the work in real-time distributed, embedded, and real-time Java has focused on the Real-Time Specification for Java as the underlying base technology, and consequently many of the chapters in this book address issues with, or solve problems using, this framework.



CONTINUOUS ENTERPRISE DEVELOPMENT IN JAVA (EARLY RELEASE)

By Andrew Lee Rubinger,
Dan Allen, and Aslak Knutsen
O'Reilly Media (September
2012)

This practical book shows you how to perform continuous development, using a testing platform called Arquillian that the authors built with the JBoss community. This platform acts as the missing link between testing and development. The authors demonstrate how testing is the very foundation of development—essential for learning and critical for ensuring that code is consumable, complete, and correct. Find out how Arquillian helps you document your project in the form of test cases.



DEVOPS FOR DEVELOPERS

By Michael Hüttermann
Apress (September 2012)
DevOps for Developers
delivers a practical, thorough introduction to approaches, processes, and tools to foster collaboration between software development and operations. Efforts of agile software development often end at the transition phase from development to operations. This book covers the delivery of software—"the last mile"—with lean practices for shipping the software to production and making it available to end users, together with the integration of operations with earlier project phases (elaboration, construction, and transition).



JAVA EE DEVELOPMENT WITH ECLIPSE

By Deepak Vohra
Packt (December 2012)

Java EE Development with Eclipse is a practical guide for using the most-commonly-used Java EE technologies and frameworks in the Eclipse integrated development environment (IDE). The book focuses on developing applications with some of these technologies using the project facets in Eclipse 3.7 and its enhancement Oracle Enterprise Pack for Eclipse 12c. It starts with a discussion of Enterprise JavaBeans (EJB) 3.0 database persistence with Oracle WebLogic Server and Oracle Database, Express Edition.

Credit Suisse Discusses the Java Community Process

We continue our series of interviews with distinguished members of the Executive Committee of the Java Community Process (JCP) with Credit Suisse, a financial services group of companies with headquarters in Switzerland and operations in more than 50 countries. Credit Suisse is the first nontechnology vendor on the Executive Committee of the JCP, elected in June 2010 and re-elected in November 2012. Credit Suisse is what [Mike Milinkovich](#) of the Eclipse Foundation calls a “Java user company” whose focus is to improve the Java platform in order to meet both its customers’ and its own business needs. As such, the company brings to the JCP a unique and valued perspective.

A woman with blonde hair and glasses, wearing a grey blazer over a black dress, walks on a city sidewalk. She is holding a small blue object in her right hand. Next to her, a man with glasses, wearing a white button-down shirt and dark trousers, walks towards the camera. He is holding a dark blue folder or book in his left hand. They are walking under a large, arched metal structure that spans the street. The background shows tall city buildings and a red sign that reads "NO STOPPING ANYTIME".

THE CHALLENGE

“Trying to build consensus is difficult because people have individual ideas about best practices... uncovering all of the use cases and coming up with a robust implementation is the overriding goal.”

sure. TCK [Technology Compatibility Kit] testing process results will be investigated; currently, the public is rarely aware of the results of the TCK testing process. All of these developments are designed to result in a more public, open, accessible, and transparent JCP.

JCP.next.2, JSR 355, JCP Executive Committee Merge, addresses the merging of the two Executive Committees (Java SE/EE and Java ME) and thus lays the foundation for one platform.

JCP.next.3, JSR 358, A Major Revision of the Java Community Process, revises the JSPA [Java Specification Participation Agreement]. This JSR will make changes to the JSPA, the process document, and the Executive Committee's standing rules with the goals of further improving the organization's processes, correcting problems that have become apparent over recent years, and clarifying language to reduce ambiguity.

Java Magazine: What structural changes would you like to see in the JCP?

Credit Suisse: Openness, transparent licensing models, ease of participation, broad participation of the Java community, and awareness of the importance of technology governance.

Java Magazine: Has Oracle delivered on

the promise of increased transparency and openness in the JCP?

Credit Suisse: Yes. Oracle is a member of the Executive Committee and has agreed and committed to JCP.next.1 and JCP.next.2 and collaborates on JCP.next.3.

Java Magazine: How could Oracle have increased participation in the JCP?

Credit Suisse: The inclusion of the JUGs in the Executive Committee is a great success, because it raises the awareness at the right level.

Java Magazine: Are there ways that the JCP could better serve the Java community?

Credit Suisse: The JCP drives the evolution of a programming language and its ecosystems and has become a very important asset for the Java community. By relating the needs of the users with those of the technology providers, the JCP can ensure that Java continues to be one of the top programming languages. We should also not forget about the next generation, and make sure that faculty and students continue to see the value of Java.

Java Magazine: What is your biggest complaint about the JCP?

Credit Suisse: The Java community is great and very active, but not all users of Java technology are aware of the importance and impact of technology governance. The JSRs are developed by the JCP community, and so perhaps we can do better at promoting ourselves to include more people and compa-

nies to engage in the evolution of Java technology.

Java Magazine: There seems to be a risk that members of Expert Groups in the JCP can be subtly influenced by their own use cases, which suggests that there is a challenge in remaining objective about what a specification needs and does not need. How do you filter out any biases you might have related to the influence of your own use cases?

Credit Suisse: Trying to build consensus is difficult because people have individual ideas about best practices, which at times may conflict. There is some ego involved in this, too, but uncovering all of the use cases and coming up with a robust implementation is the overriding goal. All spec leads have similar challenges, including finding the time while still managing their day jobs. Also, even when you find the time, it is difficult to get the other members involved, because they also have their own day jobs.

Java Magazine: Do you have any suggested improvements for Java.net as it relates to leading a JSR group?

Credit Suisse: Java.net is the JSR spec lead's best friend. It provides a wiki, source control, a mail list, archives, really everything you need to manage a JSR. We are very impressed with the fast feedback that the JCP provides for support issues. It would be great if some performance-based incentives could be provided to the Expert

Globalize Your Business



Melissa Data can help you globalize your applications as you expand operations to other countries or reach new customers in emerging markets. As a world leading data quality vendor, we offer solutions to verify, correct and standardize addresses in over 240 countries. Eliminate returns, cut postage expenses, prevent fraud and keep your customers happy by verifying their address before you send a package.

- Reduce address correction fees – save up to \$10 per package
- Efficiently validate and correct addresses every time you ship
- Maintain high customer satisfaction

www.MelissaData.com/global
or call 1-800-MELISSA (634-4772)

Accurate data. Delivered.

- | | |
|------------------------|----------------------|
| ✓ Address Verification | ✓ IP Location |
| ✓ ID Verification | ✓ Name Parsing |
| ✓ Email Verification | ✓ Phone Verification |
| ✓ GeoCoding | ✓ Record Matching |

MELISSA DATA[®]
Your Partner in Data Quality

EMBEDDED EVERYWHERE

A man with dark hair, wearing a red polo shirt with the Java logo, is pointing his right index finger towards a futuristic cityscape. The cityscape features several tall, modern buildings with blue and gold facades. Five yellow dots are placed on different buildings in the cityscape. The background is a light blue gradient with faint, large-scale patterns. The overall scene suggests a vision of a smart, connected future.

While number estimates might vary, it's clear that within this decade we will see tens of billions of new embedded computational devices entering our everyday lives—including connected vehicles, smart appliances, smart vending machines, smart meters, medical sensors, industrial controllers, and more. And Java's write once, run anywhere technology is positioned for this coming wave of diverse embedded

ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY TON HENDRIKS

ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY TON HENDRIKS



SUPER VIRTUALITY

“Because Java, by its nature, is a virtual machine [VM] environment and a virtual application platform, you can start developing and testing large parts of your application on a virtual environment—on the desktop, for example—before the embedded hardware is available.”

devices—offering an underlying platform and infrastructure that include open standards, security, reliability, scalability, sophisticated toolsets, and a 9-million-plus developer community.

Java Magazine sat down with Terrence Barr, senior technologist and product manager at Oracle, to explore the Internet of Things, and Java's role in a technology transition that some are calling a third IT revolution.

Java Magazine: What is the advantage of using Java in the embedded space?

Barr: The embedded space today is very fragmented. You have to piece together the entire stack—the run-time, the OS, the tools, the languages, the protocols, and the connectivity. We strongly believe that until we have a standards-based horizontal platform, the M2M [machine-to-machine] market won't take off, because too much

time is spent reinventing the wheel
and struggling with fragmentation
rather than building solutions.

Java is a platform and a technology that, by design, abstracts from the underlying hardware and OS, and provides a rich application environment, from very small devices to enterprise-class servers. Java is uniquely positioned to address many of these challenges because, essentially, we solved these problems 10 or 15 years ago. We solved transitions from 8- to 16- to 32-bit that the embedded space is facing today. We have a proven and secure runtime environment. We have infield updatability of application components. And Java already has a technology ecosystem with 9 million developers. They might not currently be embedded developers, but they're very familiar with the programming language and the toolset.

Java Magazine: So Java not only facilitates development in the embedded space but also acts as a *catalyst* toward embedded industry growth?

Barr: Exactly. An Oracle technology partner recently drew a parallel between the current embedded space and the early cell phone space. The whole ecosystem was held back by the fact that there was no standard platform that people could build upon. But as soon as fairly predictable phone platforms became accessible, you had this explosion of applications, accessories, hardware, and services that built upon the platforms. We see Java in the embedded space playing a very similar role, but with even more focus on creating an open, standardized technology infrastructure.

Java Magazine: What are the time-to-market advantages of developing embedded applications via desktop emulation?

Barr: Because Java, by its nature, is a virtual machine [VM] environment and a virtual application platform, you can start developing and testing large parts of your application on a virtual environment—on the desktop, for example—before the embedded hardware is available. So you can parallelize the embedded hardware development and the embedded software development, which potentially shaves many months off of your development cycle.

Java Magazine: What are the issues regarding pushing down new logic to



Barr shows off an application running on a Keil board at Devoxx.

Barr: Suppose you have millions of smart meters in the field, running the software, collecting the data, and sending the data to the back end. But at some point, the utility laws and regulations change, and you need to update the algorithms on these meters. It's not plausible to send people into the field to manually update millions of devices. Embedded hardware is getting more and more powerful, and at the same time connectivity options are increasing, which allows for wireless updates. Java offers the ability to update in the field without affecting the integrity of the system. Because Java is a virtual environment and runs in a sandbox model, it helps guarantee the security and integrity of the system after

Whether it's utility meters, automobiles, medical devices, or vending machines, all have similar requirements—you need to execute applications, and you need connectivity, security, a way to manage the device, a way to update the device, and a way for the device to connect back to the server. Delivering embedded Java as a horizontal and integrative solution provides 80 percent of your underlying infrastructure. You can then build 20 percent on top of it, adding your particular industry value. You get to market much quicker and at a lower price point.

Barr: As I mentioned earlier, with traditional native embedded development, you have to pick and choose the various components, integrate them yourself, and deal with a range of tools and methodologies. By comparison, the Java runtime is a complete development stack that is tested, is based on standards, and uses standardized APIs that you're already familiar with. So it's very easy to get started and write application and business logic. The new Oracle Java ME Embedded 3.2 not only offers a variety of standard APIs—things like XML processing, location-based services, and file access. We've

“Delivering embedded Java as a horizontal and integrative solution provides 80 percent of your underlying infrastructure. . . . You get to market much quicker and at a lower price point.”

And finally, you can use your same familiar development tools, such as NetBeans and Eclipse. And you can do live, on-device source-level debugging using these IDEs. For most Java developers, that's nothing new. But for native embedded developers, this is heaven. For them, debugging and testing is typically a very painful process.

Barr: If you have experience in the embedded space, you know that there

“One of the big values of having Java in the embedded space is the ability to bring the expertise, the tools, and the code from one platform to another—the inherent ability to apply your expertise across this whole range of devices, from very small to very large.”


The Keil board is mainly for evaluation purposes. If you want a deployment-

Java Magazine: The timeline for Oracle's embedded offerings details a 2014 JVM [Java Virtual Machine] convergence for Oracle Java SE Embedded 8 and Oracle Java ME Embedded 8. What will this mean for embedded developers and partners?


Barr: One of the big values of having Java in the embedded space is the ability to bring the expertise, the tools, and the code from one platform to another—the inherent ability to apply your expertise across this whole range of devices, from very small to very large. So about 12 to 18 months ago, Oracle embarked on a plan to converge and make more aligned the two relevant Java platforms, Java SE and Java ME.

Historically, Java ME was somewhat the smaller sibling of Java SE. But it had different features and APIs, which made it difficult to take your code and expertise and switch platforms. This strategy of aligning the two platforms

A man with dark hair, wearing a red polo shirt, is captured mid-speech. He is gesturing with his hands, which are in the foreground, slightly out of focus. The background is dark with the 'DEVVOX' logo and the text 'the java™ community conference' repeated. The logo features a stylized yellow and white icon of a mobile phone with signal waves.

 Barr talks about embedded while onsite at Devovx.



 Barr talks with Stephen Chin about embedded Java and demos a smart solar tracking system.

Barr: The CDC traditionally occupies the midrange embedded space and has been very successful in vertical product spaces—for example, every Blu-ray player ships with a CDC run-time. But we want to map the entire embedded space. So CDC will basically become a configuration of Oracle Java SE Embedded, and Oracle Java ME Embedded will cover everything else. As a developer, you'll basically look at your problem domain and your device requirements, and you'll just say, "OK, does it fit a Java ME Embedded

“What you want to do is
use the intelligence of the
local device to program-in
business logic.”

What you need to do is to turn large amounts of data into small amounts of valuable information. If you have a sensor that's monitoring the temperature in a truck, you don't want that device to just blindly send a temperature reading to a back-end server every second. That's not useful information, and it generates enormous quantities

Oracle Java Embedded Suite 7.0 provides an enterprise-like suite of technologies for the Oracle Java SE Embedded 7 runtime—Java DB relational database; GlassFish Servlet-based application server; and Oracle’s Jersey implementation of [JSR 311](#), the Java API for RESTful Web Services (JAX-RS).

What you want to do is use the intelligence of the local device to program-in business logic. If you're transporting a certain type of product that has specific temperature require-

Java brings great value to all of these processing tiers. Depending on the given solution, you can choose where and how to implement filtering and business logic. And with Java, it's always the same language, the same programming model, and the same tools and expertise. That's a huge

Java Magazine: This year at JavaOne 2012, Oracle presented an entire sub-conference called Java Embedded @ JavaOne.

During the conference, I spoke to a lot of partners and developers, and the

Steve Meloan is a former C/UNIX software developer who has covered the Web and the internet for such publications as *Wired*, *Rolling Stone*, *Playboy*, *SF Weekly*, and the *San Francisco Examiner*. He recently published a science-adventure novel, *The Shroud*, and regularly contributes to *The Huffington Post*.

- [Java Embedded](#)
- [Java Embedded downloads](#)
- [Oracle Java ME Embedded 3.2 FAQ](#)
- [Terrence Barr's blog](#)

“Many cellular networks today already might be overwhelmed in handling the data from smartphones, so how are we going to handle data from **billions of additional connected devices?**”

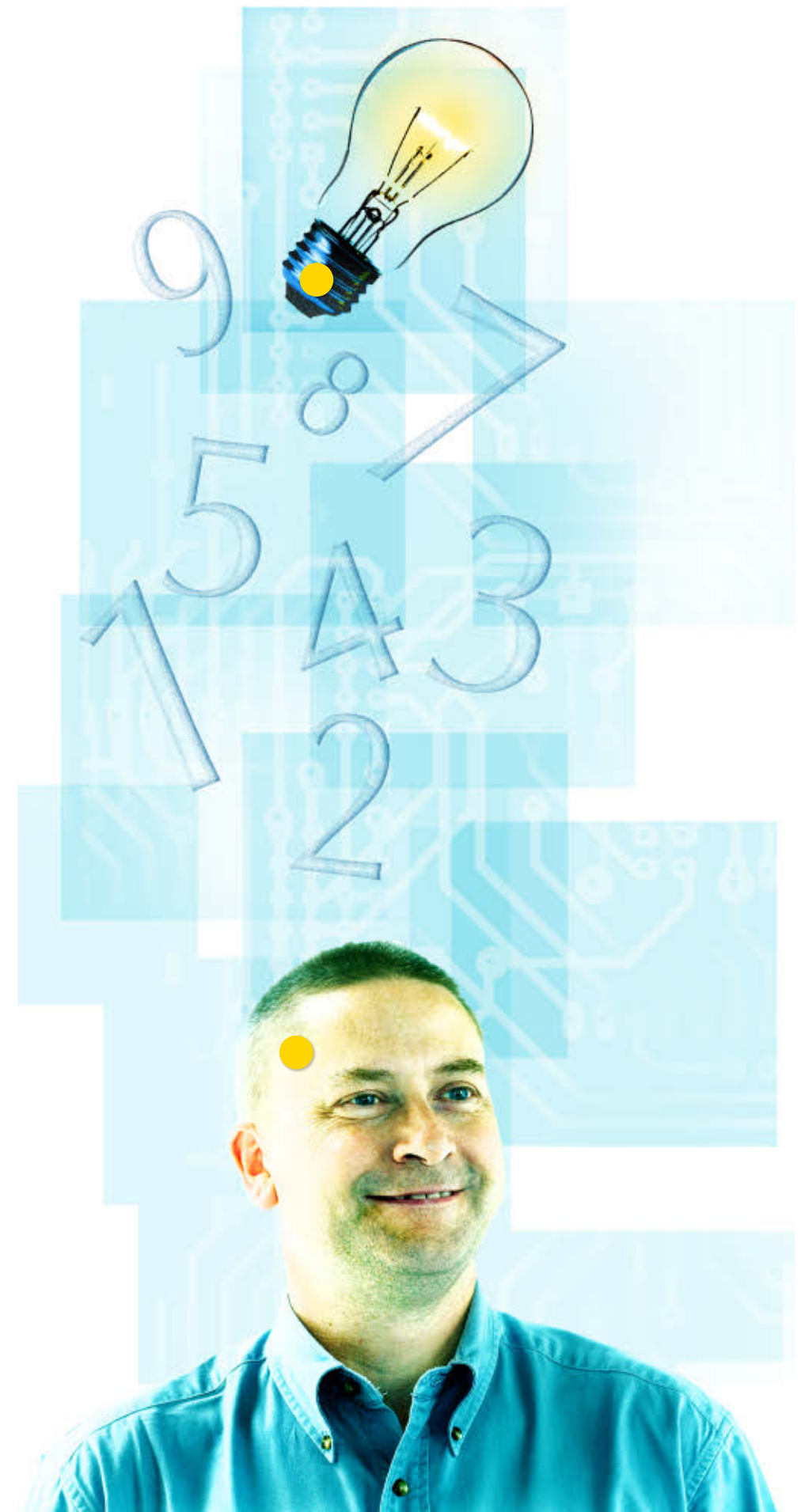
(embedded)

TOP TEN REASONS FOR USING JAVA IN EMBEDDED APPS

Learn why Java is the best language
for the Internet of Things.

BY SIMON RITTER

For many years now, we've been promised the Internet of Things—with everything from light-bulbs to cars all networked together with built-in "intelligence." The combination of Moore's Law and economies of scale is finally making this a reality. For example, Philips recently announced that it would be selling internet-enabled lightbulbs through the Apple store. For these networked, programmable devices to be really useful, they need applications to collect, process, and transmit data. Traditionally, code for embedded systems has been written in assembler or C, but for the rapid explosion of programmable, connected devices, we need something better: Java. What makes Java the best language for embedded devices? Here are ten great reasons (with thanks to [Roger Brinkley's blog entry](#) for inspiration):



1

ONE FOR ALL
Porting code in Java is a non-event. As long as none of the APIs used by the application have changed, it's simply a matter of redeploying the existing class or JAR files.

Peripheral Interface (SPI) and Inter-Integrated Circuit (I2C). However, Java's API power doesn't stop with the classes available as part of the platform. Because Java is such a popular language, developers have created libraries that address almost anything you can think of: sensors, serial port commu-

2

2

3

3

4

4

Memory management is all handled automatically by the JVM; memory is allocated by instantiating an object, rather than having to use explicit calls to library functions such as `malloc`. Developers don't have to keep track of object references and explicitly de-allocate memory either; the garbage collector handles all this. This substantially reduces the potential for memory leaks, which can have a far greater impact on embedded systems where applications might need to run for very long periods in memory-constrained

DON'T GO NATIVE
Some people wrongly think that by using a VM rather than native instructions, performance will be compromised.

environments without
needing to be restarted.

Concurrency support for applications is also handled by the JVM. The ability to create and synchronize different threads of execution has been built in from the very beginning with Java. Even if the embedded platform on which the JVM is deployed does not support multi-threading directly (some-

thing that's increasingly rare these days), it's still possible to emulate the functionality through the concept of green threads.

Some people wrongly think that by using a virtual machine (VM) rather than native instructions, performance will be compromised. With nearly 20 years of development in the JVM, these issues are no longer a significant factor. In certain situations, having precise knowledge, which is available only as the application is running, can lead to *better* performance than the performance of natively compiled code.

5 PICK AN EMBEDDED PLATFORM—ANY EMBEDDED PLATFORM.

Embedded systems are generally designed to solve a specific problem, whether it is how to provide in-car entertainment or how to monitor the

pH level of part of an industrial process. As such, each system will have hardware tailored to the solution being developed, which gives designers far greater choice than when creating a new desktop or laptop system.

Many embedded chip manufacturers, such as Freescale and Broadcom, create processors and systems on a chip (SoCs) using architectures from companies such as ARM. Although this makes certain aspects of the different processors and SoCs standard, there are often small differences in terms of aspects such as floating-point processing, bus architectures, and so on. Java abstracts these differences away from the developers, making their lives significantly easier.

Reference implementations of the Oracle Java Embedded Client are available for Intel x86 and MIPS as well as ARM v5, v6, and v7 architectures running Linux (the most popular OS for embedded devices). Oracle Java SE Embedded is available for Intel x86 and IBM Power e500v2 and e600 systems as well as ARM v5, v6, and v7 architectures, again running Linux. This gives embedded systems designers plenty of choice about which platform to use.

6 THINGS HAVE CHANGED SINCE 1995.

Java was originally developed in the early 1990s to allow applications to be written for the Star7 PDA, which was an embedded device. When Java was first launched as a general-purpose computing platform in 1995, the average personal computer had 8 MB of RAM and a processor running at less than 200 MHz. Even today, the full Java SE runtime requires only a minimum of 64 MB of RAM to run on Windows XP. Typical low-end embedded systems now match or exceed these specifications.

Java was developed from the very beginning to work in resource-constrained environments, making it ideally suited to the needs of embedded systems today. Oracle Java ME Embedded will run in as little as 350 KB of ROM and 130 KB of RAM (look carefully, that's *kilobytes*, not *megabytes*), growing to 1.5 MB of ROM with 700 KB of RAM for the full, standard configuration. Oracle Java SE Embedded is designed for more-powerful systems, but still requires only 39 MB of ROM and 32 MB of RAM.

Having to deal with lower clock speeds also means that the Java platform works well when you want to maximize the time between charges for battery-powered devices. When resources are tight, Java can still get the job done.

8

DEVELOP ON ONE MACHINE, DEPLOY TO ANOTHER.

Embedded systems by their very nature tend not to resemble desktop or laptop systems. Frequently they are what is called *headless*, meaning that they have no display attached. This adds extra challenges and complexity to software development in terms of how to compile the code on one machine and deploy it to another.

Because Java uses a VM, it doesn't matter where you compile your code, so there's no need to set up complex cross-compilation tool chains. Use your favorite Java IDE to develop the code in comfort on your desktop, and then simply copy the class files or JAR files to the target device. To make developers' lives even easier, most IDEs support the concept of remote debugging through a network connection. If the code is doing what you told it to do, rather than what you wanted it to do, simply use the debugging facilities in your IDE to figure out what the issues are. This will be a lot faster than using print statements.

9

NINE MILLION DEVELOPERS
CAN'T BE WRONG.

Depending on whose survey you consult, there are up to 9 million developers in the world who know how to

program in Java. Almost all universities use Java as a teaching language for object-oriented programming fundamentals, which means that the number of Java developers is likely to increase.

If you want to develop Java code, you have a huge pool of programming talent to draw on. Because the JVM and the class libraries abstract away much of the complexity of developing embedded system code, you don't need to find people who have lots of experience in this field. For situations where more-domain-specific knowledge is required, there will still be a larger number of Java-capable developers to fill these positions.

10

FROM DEVICE
TO DATA CENTER...

For embedded systems to have real value, they need to be networked, not just to other embedded systems but also to places where data can be aggregated, analyzed, and searched. This is what data centers are good at, recording and processing huge amounts of data from many different sources and then mining the data for

BIG TALENT POOL

Because the JVM and the class libraries abstract away much of the complexity of developing embedded system code, you don't need to find people who have lots of experience.

useful information. What is the most popular platform for developing enterprise applications? Java. Using Java in embedded devices, in the data center for big data processing, and on the desktop for presenting and controlling the whole system makes architecting a complete solution simpler, quicker, and cheaper.

As you can see, Java offers benefits for developing code for embedded systems. With lower cost, more choices of platform, and readily available skilled developers, there's nothing holding you back

from helping to build the Internet of Things. </article>

Simon Ritter is a Java technology evangelist at Oracle. He has worked in both Java technology development and consultancy. He now focuses on the core Java platform and Java for client applications.



32

Get Started with Raspberry Pi

Get Started with Raspberry Pi

If you are ready to dive into programming on the Raspberry Pi, don't miss the article "[Get Started with Java SE for Embedded Devices on Raspberry Pi](#)," in the November/December 2012 issue of *Java Magazine*. Authors Bill Courington and Gary Collins provide an in-depth, step-by-step guide to getting Linux and Oracle Java SE Embedded running on your Raspberry Pi in less than an hour.

Broadcom Corporation, where Upton is employed as an SoC (system-on-a-chip) architect. In 2010, he was involved in the design and development of the Broadcom BCM2835 multimedia applications processor.

"Having decent graphics performance and multimedia features can attract children to the platform," Upton says. "Gaming and Blu-ray-quality movie playback are hooks to draw young users to the device."

FINDING THE SWEET SPOT

While 512 MB of RAM may seem prohibitively small for a programming environment, Java actually fits well within those confines. "Java is a language that is ideal for creating small applications that run very portably," says Simon Ritter, technology evangelist at Oracle. "The Java devel-

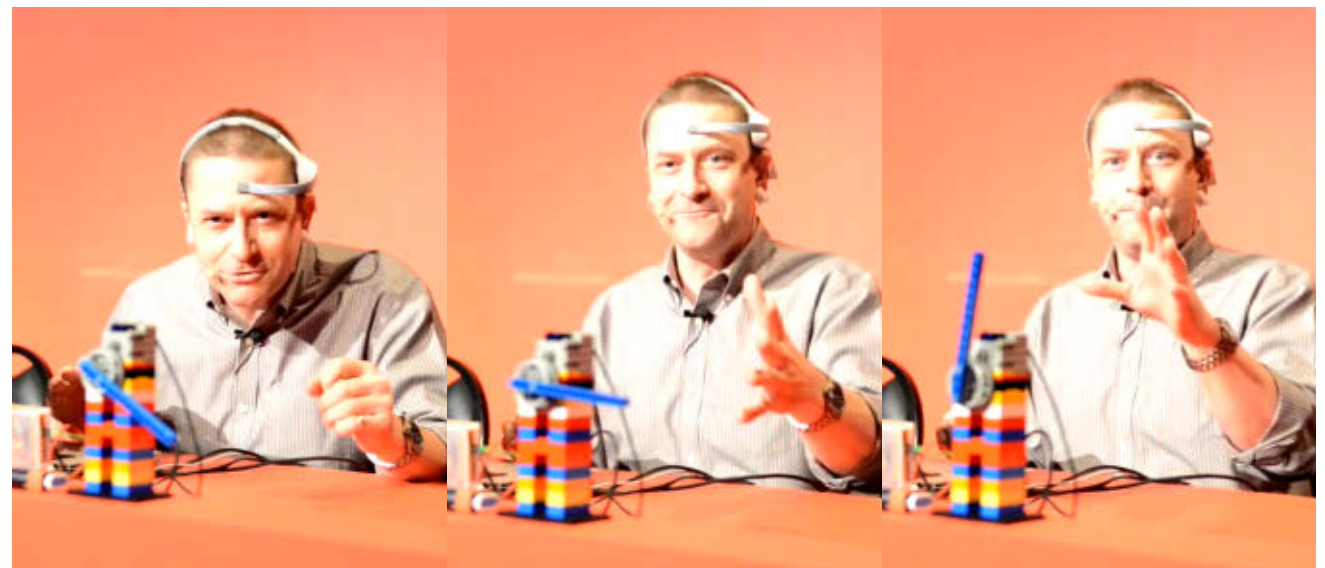
opment community is accustomed to these kinds of limitations. When Java was created back in the 1990s, most desktops had only 4 or 8 megabytes of memory.”


Optimizing Java for the tiny computer's ARMv6 processor requires some finesse. The challenge is related to floating-point arithmetic—the way the processor performs arithmetic on real, or nondecimal, numbers. Many chips based on the ARMv6 architecture lack hardware floating-point units; the Broadcom BCM2835, however, does include ARM's virtual floating point (VFP) coprocessor.

A port of the JVM that uses soft floating point on ARM processors is currently available; it will work on the Raspberry Pi and is supported by Oracle. Oracle is creating a different

version of the JVM that will run on the Raspberry Pi's processor and will take advantage of the acceleration for hard floating-point arithmetic. That should yield a 10 to 15 percent improvement in performance for more-compute-intensive applications such as those that use graphics. "We're in a comprehensive testing cycle for the optimized version of JVM for the Raspberry Pi," Ritter says. "That should deliver a lot of benefits for users."

What else does Java bring to this tiny computer? For Upton, it all comes back to educating children, the objective at the heart of the Raspberry Pi Foundation. "There are some high-quality educational applications that run on Java that we can bring across with no investment," he says. "That's very important to us."



 Simon Ritter demonstrates mind control of a Lego windmill using only his brain (and a Raspberry Pi running Java).

Of course, ultimately these students will move into the business world, and some of them will use their skills

Upton also recognizes that his invention will soon expand beyond the classroom. "We're going to create a lot of value for people developing industrial applications," he says, pointing out that a small, affordable general-purpose computer could ultimately replace a wide range of specialized devices, especially those that require embedded microprocessors to perform essential tasks on the shop floor, on an assembly line, and for finely tuned scientific applications.

LOOKING AHEAD

Future enhancements may include Wi-Fi plug-and-play with firmware revisions, an onboard Skype service, and add-on expansion boards for cameras.

"Once hardware is a given, people will be able to innovate at the software level," Upton concludes. "We are very hopeful that the Raspberry Pi and its educational initiatives will refill the pipeline of individuals who will shape the of us." [</article>](#)

Based in Santa Barbara, California, **David Baum** and **Ed Baum** write about innovative businesses, emerging technologies, and compelling lifestyles.



Oracle is in the process of porting the JVM to run on the Raspberry Pi's ARMv6 700 MHz processor.



Brûlé and Everett surround themselves with coins at Royal Canadian Mint headquarters in Ottawa.

SNAPSHOT ROYAL CANADIAN MINT

mint.ca

Headquarters:
Ottawa, Ontario,
Canada

Industry:
Government

Employees:
More than 900
in 2011

Revenue:
CA\$3.2 billion
in 2011

**Java technology
used:**
Java Card

"For years we have been following the advent of electronic payments and their growth," Brûlé continues. "Three or four years ago, we decided to dedicate some research and development funding to looking into electronic payments and, more specifically, what needs in the marketplace weren't being fulfilled and whether there was a role the Mint might play in this emerging market."

From that decision came a new digital currency, prototypes of which were introduced last year: the MintChip. Many forms of digital currency already exist—from electronic funds transfer (EFT) to direct deposit and loyalty cards—but the RCM sees the MintChip as a potential replacement for physical currency, both coins and paper, in

market. It offers a good value proposition for merchants and consumers alike because it doesn't run across a network, is highly portable, and provides chip-to-chip secure communications."

The MintChip is built entirely on Java Card technology, the de facto standard platform for smartcards and other small memory footprint devices. More than 12 billion Java Card-enabled devices have shipped since the platform was introduced in 1997, according to Brian Kowal, Java Card business manager at Oracle. "Java Card is the world's number one leading software application platform by volume," says Kowal. (See sidebar, "A Pocketful of Java Card.")

Kowal says the MintChip has one advantage that none of its rivals thus

Canada and other countries around the world. Being digital, says Brûlé, the MintChip can be denominated in almost any world currency.

"We see the MintChip as being an option to physical cash," says Brûlé. "We believe the MintChip can be very cost-effective in the low-value transactions mar-

far have: "The MintChip is the only digital currency in the world being researched by a sovereign government," says Kowal. Given that the Canadian dollar is one of the world's most widely traded currencies, that adds substantial credibility.

POCKET CHANGE

The RCM was founded in 1908 as a branch of the UK Royal Mint and in 1969 was incorporated as a for-profit commercial crown corporation. In monarchical Commonwealth countries such as Canada, crown corporations are government-owned enterprises that undertake activities on behalf of their sole shareholder—in the RCM's case, the Canadian government. While the Bank of Canada, a separate crown corporation, issues all Canadian banknotes, the RCM mints the "pocket change" Canadians use every day to buy a cup of coffee, pay a transit fare, or purchase a newspaper or magazine. These include the "Loonie," a CA\$1 coin so nicknamed because it bears the image of a common loon on one side, and the CA\$2 "Toonie," whose nickname blends the words *two* and *loonie*.

Besides Canadian coins, the RCM also mints commemorative and collector coins, bullion coins, medals and medallions, and even coins for other sovereign governments—in any given year, the RCM contracts to manufacture coins for as many as a

Java Card was the obvious platform choice for the MintChip, says Everett, with Brûlé near the Ottawa River.

MONEY FOR THE MOBILE GENERATION

One of the target markets for the MintChip, at least initially, is younger mobile phone users in developed and emerging economies worldwide who are “unbanked”—that is, they do not have regular bank checking or savings accounts and regularly deal in cash transactions.

dozen foreign countries. "We operate like any other corporation," says Brûlé. "We operate to make a profit, and we receive no government funding."

In prototype form, the MintChip is an SD, microSD, UICC (Universal Integrated Circuit Card), or USB memory card containing an integrated circuit that provides security. It comes loaded with the MintChip application; the Java Card platform, including a Java Virtual Machine (JVM); and an operating system. Written in Java, the MintChip runs atop the JVM within the Java Card runtime, so there's nothing to stop it from being deployed in the cloud as well, adds Brûlé.

All Java Card platform chips—including those the MintChip uses—are highly secure, says Oracle's Kowal.

They are certified under the US FIPS 140-2 and European Common Criteria EAL5+ standards, used for military and government IDs and payment systems worldwide. "What makes this interesting," says Kowal, "is that the MintChip could be embedded in an automobile, a smart meter, or a personal e-reader or tablet and provide payment solutions for all a consumer's personal connected devices."

“Java Card hardware has unique attributes to secure against physical attacks,” continues Kowal. “They have small memory footprints, are low-power, and have a persistent memory model that holds transactions even if power is removed midway.”

One of the target markets for the MintChip, at least initially, is younger

mobile phone users in developed and emerging economies worldwide who are “unbanked”—that is, they do not have regular bank checking or savings accounts and regularly deal in cash transactions.

“The way the MintChip works is what we call an asset transfer model,” explains Dr. David Everett, an independent consultant and the MintChip’s chief technical architect. “The consumer actually holds the value in their chip. If you have a chip in your phone, for instance, the value’s actually in that chip. And when you make a payment to somebody else, it leaves your chip and goes to another chip.”

A typical use case for the MintChip would be mass transit, Everett says. Consumers load their MintChip



Use Mutual Certificates Security to set up trusted authentication and protection that ensures the integrity and confidentiality of messages.

- In [Part 1](#), we explored Username Authentication with Symmetric Key to generate a certificate/key couple shared by the client and the server to sign and encrypt messages. We also used the username/password combination to authenticate the client during service calls.
- In [Part 2](#), we explored how to protect the data transiting between the client and the server using Secure Sockets Layer (SSL) via the secure HTTP transport, HTTPS.

Note: The complete source code for the application designed in this article can be downloaded [here](#).

Mutual certification, also called *shared certification*, is a process or technology where two communicating entities authenticate mutually. The Mutual Certificates Security (MCS) mechanism provides security through authentication and message protection to ensure integrity and confidentiality. This mechanism

The server certificate is provided to the client in advance of any communication, so both the client and the server can be sure they are dealing with legitimate entities, as shown in **Figure 1**.

Download the following software, which was used to develop the application described in this article:

- NetBeans IDE 7.2 (available for download [here](#))
- GlassFish 3.1.2.2 (available for download [here](#))
- Metro 1.3 or higher (included in NetBeans)



Note: This article was tested using the latest version of the NetBeans IDE (version 7.2, at the time this article was written).

Overview of Adding MCS to the Web Service

What we are going to do is secure both sides of the AuctionApp application by using the MCS mechanism. Specifically, we will

configure the keystore and the truststore for the client and the server to ensure the integrity and confidentiality of the message.

We will perform the following tasks:

- Secure the Web service by adding the MCS mechanism and configuring the keystore and the truststore for the server
- Create and configure a new

Web client that references the Web service by specifying the secure Web Services Description Language (WSDL) file

- Test the noncertified client application
- Configure the keystore and the truststore for the client
- Test the certified client application

Note: The application we are going to secure is an online auction place (like eBay) that we created in a previous series of articles (["Introduction to RESTful Web Services"](#)). Sellers post their items in listings, and buyers bid on the items. A seller can post one or many items, and a buyer can bid on one or many items.

Specifically, we are going to limit access to the Java API for XML Web Services (JAX-WS) Web service that extrapolates the amount of a bid by multiplying by a factor (100).

Add MCS to the Web Service

It will take less than five minutes to add MCS to the Web service using NetBeans IDE 7.2.

1. Add the MCS security mechanism to the AuctionApp application:
 - a. Open the AuctionApp project in NetBeans IDE 7.2 or higher.
 - b. Expand the **Web Service** node of the AuctionApp project and right-click the

AuctionAppSOAPws node;
then select **Edit Web Service**
Attributes.

- c. Under the Quality Of Service tab, expand the **AuctionApp SOAPwsPortBinding** section.
 - d. Make sure the **Reliable Messaging Delivery** option is *deselected*.
 - e. Select **Secure Service** and then select **Mutual Certificates Security** from the Security Mechanism list, as shown in **Figure 2**.
2. Configure the keystore to specify the alias identifying the server certificate and private key for the AuctionApp application:
- a. Deselect **Use Development Defaults** if it is selected.
 - b. Click the **Keystore** button, and then click the **Load Aliases** button and select **xws-security-server**, as shown in **Figure 3**.
 - c. Click **OK**.

At this point, NetBeans creates a new Web Services Interoperability Technologies (WSIT) configuration that contains the security elements within the `sc:KeyStore` tags of the `wsit-com.bonbhel.oracle.auctionApp.AuctionAppSOAPws.xml` file, as shown in **Figure 4**. The file is located in the Web

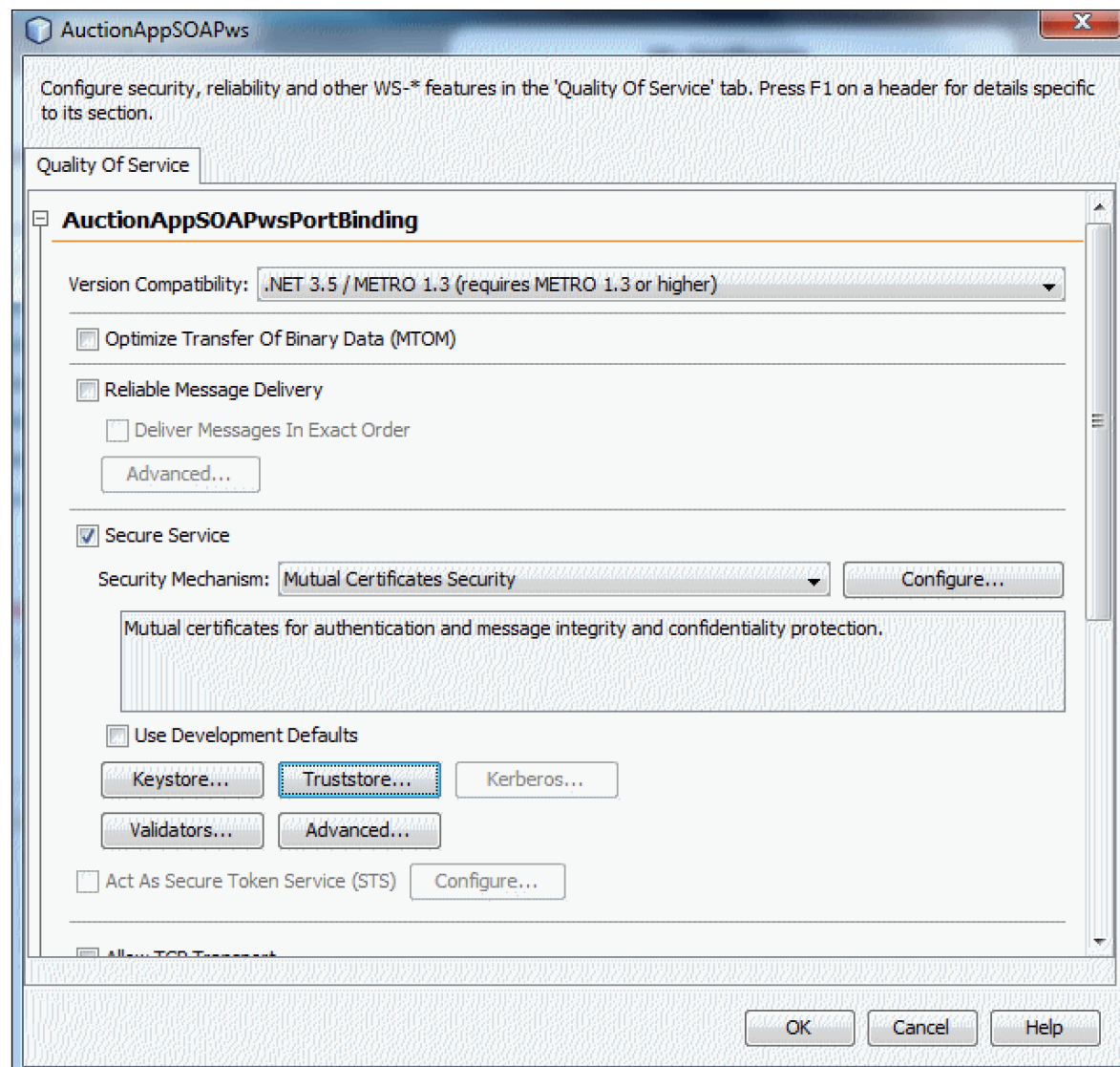


Figure 2

Pages/WEB-INF node of the AuctionApp project.

3. Deploy the service and display the WSDL file so you can inspect the `sp:AsymmetricBinding` tags:
 - a. Right-click the **AuctionApp** node and choose **Deploy**.
 - b. Open your browser and type the WSDL URL: <http://localhost:8080/AuctionApp/>

AuctionAppSOAPws?wsdl.

The Web service presents its WSDL file, as shown in

Figure 5.

The `sp:AsymmetricBinding` tags contain all we need to encrypt and sign the SOAP message using asymmetric key algorithms (public/private key combinations). This tag should contain nested elements such as the following:

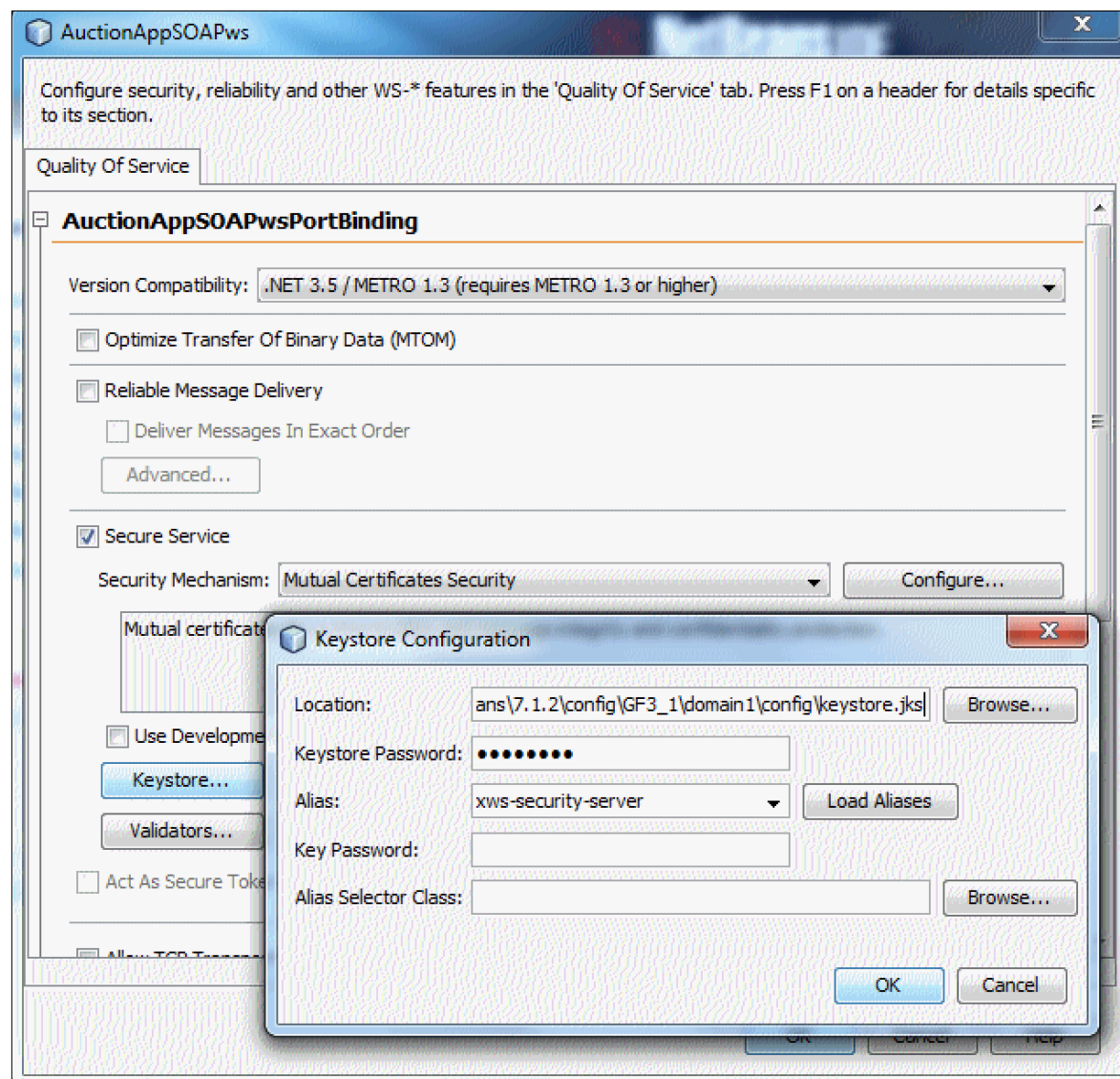


Figure 3

- **<sp:InitiatorToken>**: The *initiator token* refers to the public/private key-pair owned by the initiator.
- **<sp:X509Token>**: This tag specifies that the X.509 certificate token profile will be used. It contains the **sp:IncludeToken** attribute, which is responsible for adding the public key to the message sent to the recipient.

Create a New Web Client

In this section, we are going to create and secure a new Web service client that references the Web service that we just secured.

To do this, we will create a sample Web client application by using the Web Service Client wizard provided by NetBeans IDE 7.2 to generate the code and everything we need for looking up a Web service.



Figure 4

```

▼<sp:AsymmetricBinding>
  ▼<wsp:Policy>
    ▶<sp:AlgorithmSuite>...</sp:AlgorithmSuite>
    <sp:IncludeTimestamp/>
    ▼<sp:InitiatorToken>
      ▼<wsp:Policy>
        ▼<sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
          securitypolicy/200702/IncludeToken/AlwaysToRecipient">
          ▼<wsp:Policy>
            <sp:WssX509V3Token10/>
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:InitiatorToken>
    ▶<sp:Layout>...</sp:Layout>
    <sp:OnlySignEntireHeadersAndBody/>
    ▼<sp:RecipientToken>
      ▼<wsp:Policy>
        ▼<sp:X509Token sp:IncludeToken="http://docs.oasis-open.org/ws-sx/ws-
          securitypolicy/200702/IncludeToken/Never">
          ▼<wsp:Policy>
            <sp:RequireIssuerSerialReference/>
            <sp:WssX509V3Token10/>
          </wsp:Policy>
        </sp:X509Token>
      </wsp:Policy>
    </sp:RecipientToken>
  </wsp:Policy>
</sp:AsymmetricBinding>

```

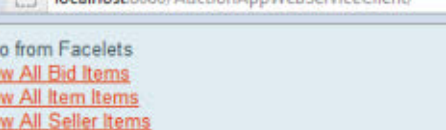
Figure 5

-
- Steps**
1. Choose File Type
 2. **WSDL and Client Location**
- WSDL and Client Location**
- Specify the WSDL file of the Web Service.
- ☐ Project:
- ☐ Local File:
- ☒ WSDL URL:
- ☐ IDE Registered:
- Specify a package name where the client java artifacts will be generated:
- Project:
- Package:
- ☐ Generate Dispatch code
-

LISTING 2 LISTING 3

 [Download all listings in this issue as text](#)

- [illegible]



localhost:8080/AuctionAppWebServiceClient/

Hello from Facelets

[Show All Bid Items](#)

[Show All Item Items](#)

[Show All Seller Items](#)

List

1.4/4

Id	Amount	Biddername	ItemId	
10	8000.0	Carolis	1	View Edit Destroy
2	10.0	Fali	1	View Edit Destroy
5	800.0	Maroc	4	View Edit Destroy
3	12.0	Vals	2	View Edit Destroy

[Create New Bid](#)

[Index](#)

View

Id: 3

Amount: 0.0

Biddername: Vals

ItemId: 2

[Destroy](#)

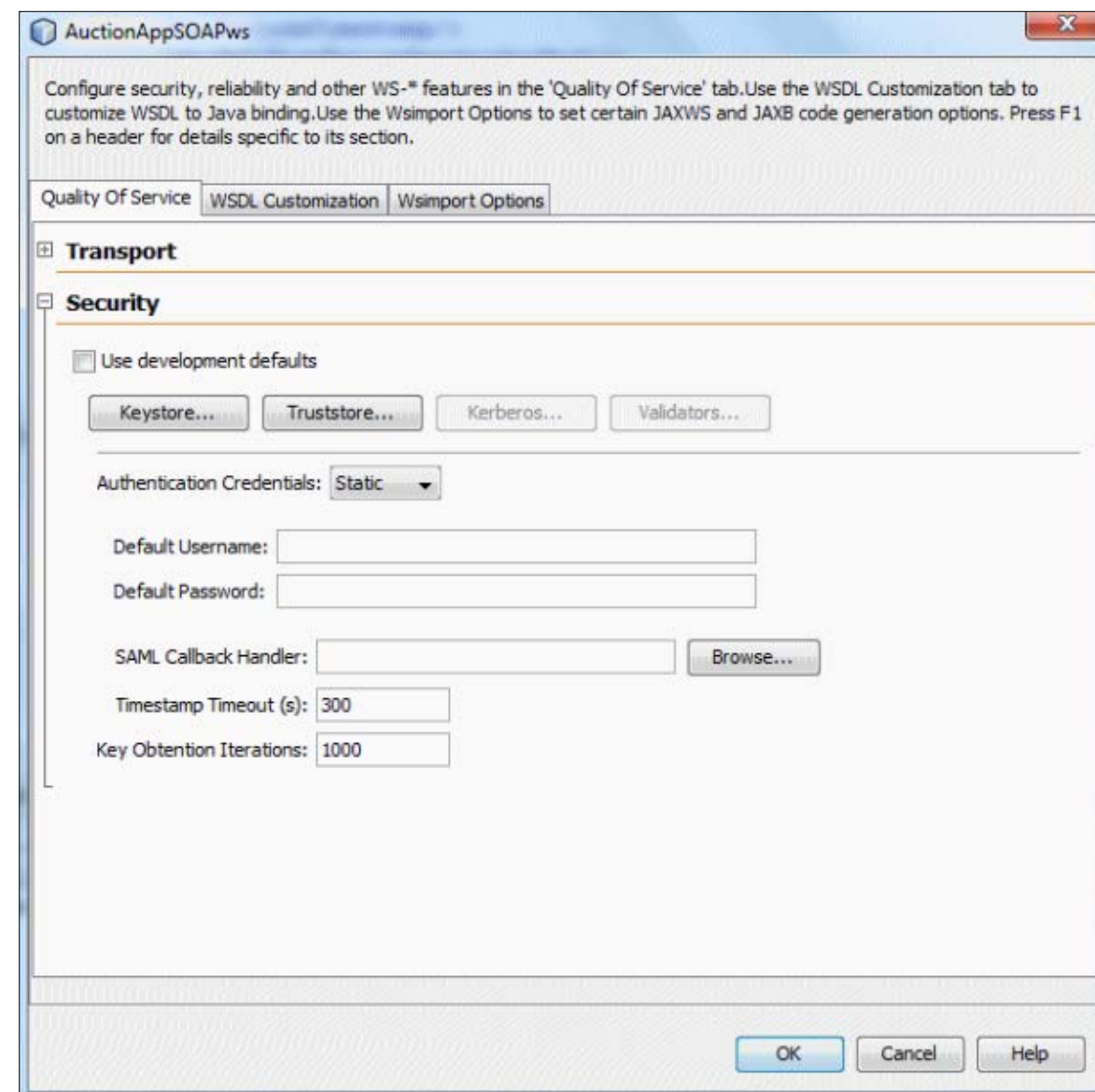
[Edit](#)

[Create New Bid](#)

[Show All Bid Items](#)

[Index](#)

1. Make sure the AuctionApp project is up and running. If it is not, right-click the **AuctionApp** node and choose **Deploy**.
2. Open the **Web Service References** node of the AuctionAppWebServiceClient project and right-click the **AuctionAppSOAPws** node; then select **Refresh**.
3. From the Confirm Client Refresh wizard, make sure **Also replace local wsdl file with original wsdl located at:** is selected; then click **Yes**.
4. Right-click the **AuctionAppWebServiceClient** project and choose **Clean and Build**.
5. Right-click the **AuctionAppWebServiceClient** project again and choose **Run**. The list of all entries is displayed, as shown in **Figure 7**.
6. Click the **Show All Bid Items** link to display the list of bid



entries, as shown in **Figure 8**.

7. Click the **View** link for the bidder named Vals to see the newly extrapolated amount of the Vals bid, as shown in **Figure 9**.

As you can see, the amount of the bid changed from 12.0 to 0.0. This means that the client failed to call the secured Web service.

- Configure the keystore to point to the alias for the client certificate
- Configure the truststore that

- Using a username/password combination to access the service



- As demonstrated, NetBeans and GlassFish make adding security for Web services easier than ever. </article>

- [NetBeans Advanced Web Service Interoperability manual](#)
- [Metro User Guide](#)
- [GlassFish resources](#)



Written by leading technology professionals, Oracle Press books offer the most definitive, complete, and up-to-date coverage of Oracle products and technologies—including the latest Java release.

Acclaimed programming author Herb Schildt's books have sold more than 3.5 million copies worldwide



Java: The Complete Reference, Eighth Edition

Herb Schildt

A fully updated edition of the definitive guide for Java programmers



Java: A Beginner's Guide, Fifth Edition

Herb Schildt

Essential Java programming skills made easy



Java Programming

Poornachandra Sarang

Learn advanced skills from an internationally renowned Java expert



Demystifying invokedynamic

The release of Java 7 brought substantial additions to the language, such as the new [try-with-resources](#) statement, multi-catch clauses, and the diamond operator. New APIs have been introduced, too, with notable examples being the fork-join framework for parallel algorithms and NIO.2 to better deal with native file system capabilities.

(JVM) bytecode instruction called **invokedynamic**. It was introduced to facilitate the implementation of dynamic languages on top of the JVM, which is an attractive target platform with a robust adaptive runtime that, in recent years, has seen many languages flourish. Some are ports of existing languages to the JVM (for example, Rhino, Jython, and JRuby) while some are new (for example, Groovy, Clojure, and Fantom).

This article is a gentle introduction to using `invokeDynamic` in your own code. The topic in itself is fairly rich and will mostly appeal to language and middle-ware implementers. We will only scratch the surface of the provided APIs and possible usages, but by the end you should have a basic technical understanding of `invokedynamic`.

Note: The source code for the examples in this article can be downloaded [here](#).

as the name, and `(Ljava/lang/String;)V` as the JVM internal representation signature. Invoking a method requires placing a *receiver* object—that is, the object on which the method shall be invoked—on the stack. The only exception is the case of static methods, which do not have a receiver.

The following are the four invoke instructions:

- **invokestatic**, which is used for static methods.
- **invokevirtual**, which is used for methods that require dynamic dispatch, that is, **public** and **protected** methods.
- **invokeinterface**, which is similar to **invokevirtual**, except that the method dispatch is based on an interface type.
- **invokespecial**, which is for the other types of methods: constructors and **private** virtual methods. The method dispatch is

The JVM has traditionally offered four instructions for invoking methods. Each points out an owner class, a name, and a signature description. As an example, given a static method `void baz(int a, String s)` in class `foo.Bar`, invoking the method requires a reference with `foo/Bar` as the owner, `baz`



PHOTOGRAPH BY
MATT BOSTOCK/GETTY IMAGES

ments; wrap arguments to or unwrap arguments from an array; perform type conversions; and more. While it is beyond the scope of this article to provide a complete overview of each of these, you will discover some commonly used ones if you keep reading.

Example of call site and target adaptation. You will often run into the problem of adapting call sites and targets when working with *invokedynamic*. Suppose that you have a method `printAll` declared as shown in **Listing 4**.

Clearly, `printAll` is of type `(String, Object[])void`. In Java, invocations of this method would call sites such as those shown in **Listing 5**.

Now suppose that we have call sites expecting something different, such as a method that takes a fixed number of parameters and no prefix string. Clearly, the signatures do not match. A fix would be to introduce an adaptation method such as that shown in **Listing 6**.

Such an adaptation is simple to perform using method handles and combinators, as shown in **Listing 7**.

First, `target` is a direct method handle to the `printAll` static method. We use the `bindTo` combinator to bind the first argument to a value. In the case of a static method, this is effectively the first argument, while in the case of an

instance method, the object would be the receiver.

Then, the `asCollector` combinator wraps arguments into a collecting array. Here, we bind four arguments into an array of `Object` instances. Using the code in **Listing 8**, we can check that the resulting method handle is of type `(Object, Object, Object, Object)void` and that invoking it with arguments of the expected arity works.

Performance-wise, you should use `invokeExact` rather than `invokeWithArguments` whenever you can. `invokeExact` requires the argument types to be exactly like those of the method handle type descriptor, and it dispatches the call much faster. In contrast, `invokeWithArguments` performs type checks and conversions.


A classic optimization for virtual methods is to build *inline caches* where the target method handle is cached as long as the receiver type remains stable. You should limit the cache depth in case the call site happens to be megamorphic; otherwise, you would create too many method handles arranged in a chain of guarded tests, which further degrades the performance figures.

Bootstrapping

At this point, you should have a basic understanding of what

LISTING 4 LISTING 5 LISTING 6 LISTING 7 LISTING 8

```
static void printAll(String prefix, Object... values) {
    for (Object value : values) {
        System.out.println(prefix + value);
    }
}
```

 [Download all listings in this issue as text](#)

method handles are. We can now turn to the bootstrapping side of *invokedynamic*.

CallSite objects. Call sites need to be bound at runtime using the `java.lang.invoke` API. To do that, each *invokedynamic* instruction refers to a *bootstrap method* that is executed the first time the call site invocation is executed.

The bootstrap method must return an instance of `java.lang.invoke.CallSite`. As the name suggests, such objects represent a call site, and they are bound to a target method handle, which may, in turn, be a chain of combinators.

The `CallSite` can be thought of as the container for the `MethodHandle`. Once it has been bound, a call site always references the same `CallSite` instance. Although the `CallSite` itself will refer to the same program point throughout its lifecycle, call sites may allow their target method handle to be redefined, provid-

ing a way to swap out old code with new code on the fly, without regenerating the bytecode for an entire method.

The `java.lang.invoke` package offers three concrete implementations of the abstract `CallSite` class, and you can subclass them, too:

- `ConstantCallSite` is for call sites whose target method handle never changes.
- `MutableCallSite` has object field semantics, which means that the target method handle can be changed.
- `VolatileCallSite` is similar to `MutableCallSite` but with *volatile* reference semantics.

If you need a call site where the target can be changed, choosing between `MutableCallSite` and `VolatileCallSite` is a matter of understanding the Java memory model and weighing the consequences of choosing one versus the other. In multithreaded environments, a *volatile call site target*

change will be immediately visible to all threads, while with ordinary field semantics, thread caching occurs and requires explicit synchronization (see the static `syncAll` method in `MutableCallSite`). Of course, there are performance implications, too.

Bootstrap methods. `invokedynamic` bootstrap instructions are bound by invoking a static method that returns a `CallSite` object that must take at least three parameters:

- A `MethodHandles.Lookup` object that is bound to the classes visible in context of the call site
- A symbolic name as a string
- A `MethodType` that corresponds to the type that is expected by the call site (this is often useful for performing a final `asType()` combinator invocation to ensure that method handles match)

Other parameters can be passed as extra arguments, with the only constraint being that they must be constant values, which means they can be written to the *constant pool* of a class bytecode representation. This is the case for primitive types, strings, class references, and method handles.

Sticking to the example in **Listing 8**, which obtained a method handle for the `printAll` method, the code in **Listing 9** would be a possible call site bootstrap method.

This method returns a non-modifiable call site, where the prefix string is bound to `">>>"` and the remaining arguments are collected as variable arguments. The resulting call site is of type `(Object[]) void`. Note that because a string

is a constant, we could also obtain the prefix as an extra parameter, and the value would depend on what is referenced from the call site in the bytecode. In this case, the bootstrap method would be as shown in **Listing 10**.

It is easy to test the bootstrap method to verify that it works as intended. Indeed, call sites provide a `dynamicInvoker` method that returns a method handle. Thus, you can use it as we've previously seen (see **Listing 11**).

Emitting Bytecode with `invokedynamic` Instructions

We are now ready to assemble the final pieces and actually put `invokedynamic` into practice. While the `java.lang.invoke` API can be

KEY FOR LAMBDAS

Java 8 is poised to take advantage of `invokedynamic` as a way to support lambdas.

LISTING 9

LISTING 10

LISTING 11

LISTING 12

```
public static CallSite bootstrap(Lookup lookup, String name,
                                MethodType type) throws Throwable {
    MethodHandle target = lookup.findStatic(
        Sample3.class, "printAll",
        methodType(void.class, String.class, Object[].class))
        .bindTo(">>> ")
        .asVarargsCollector(Object[].class)
        .asType(type);
    return new ConstantCallSite(target);
}
```



Download all listings in this issue as text

for the function can be anything, but it is common to normalize names and separate portions using a colon, as in `property:get`, `property:set`, and so on.

We can run this method, as shown in **Listing 13**, and see that the `invokedynamic` instruction is correctly bound at runtime. We can also check the generated bytecode with the `javap` decompiler that comes as part of the JDK, as shown in **Listing 14**.

Conclusion


This article introduced [invoke dynamic](#), a new instruction backed with runtime API support for facilitating the implementation and execution of dynamic languages on top of the JVM. The [java.lang.invoke](#) API provides a rich set of operations to adapt a call site to a target that might be of a different signature type.

Existing dynamic languages for the JVM (for example, Groovy and JRuby) are starting to support **invoke dynamic**. In the long run, this should reduce the footprint of their runtime support code because maintain-

BETTER SUPPORT
The introduction of invokedynamic to the JVM specification is recognition of the fact that dynamic languages bring value to the Java ecosystem and, as such, the JVM needs to provide better support for them.

LISTING 13 LISTING 14

```
$ java -classpath code/build/classes sample3.Caller
>>> World
>>> Hello
```

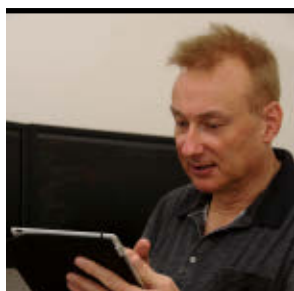
 [Download all listings in this issue as text](#)

language? In any case, there is no harm in trying, and who knows what you might come up with.

Acknowledgements. The author would like to thank Marcus Lagergren for his very constructive feedback. `</article>`

LEARN MORE

- "Bytecodes Meet Combinators: invokedynamic on the JVM"
- "JooFlux: Hijacking Java 7 InvokeDynamic to Support Live Code Modifications"
- "JSR 292 Cookbook"
- "Dyalink: Dynamic Linker Framework for Languages on the JVM"



Use a new plug-in mechanism to extend the Java compiler with new behavior.

In this article, we show how you can write a simple, customized source code analysis tool so you can learn how to leverage the plug-in mechanisms for your own applications. We find code patterns that check whether the result

- Plug-ins do not use the model of processing rounds, which incurs additional overhead.

- The `call()` method gives access to the compiler functionalities through a `JavaTask` object, which allows you to perform parsing, type checking, and compilation. In addition, the `JavaTask` object lets you add observers (which are instances of `TaskListener`)

The first step is to download and build the current release of the Java 8 compiler, which supports compiler plug-ins. Toward this end, download the latest version and fol-

Once the compiler is built, include the generated `dist/lib/classes.jar` file in your project. Alternatively, you can download a ready-made binary from jdk8.java.net.

1. Implement the `com.sun.source.util.Plugin` interface shown in **Listing 1**.
2. Add a `TaskListener` to perform additional behavior after the type-checking phase.
3. Create an abstract syntax tree (AST) visitor to locate binary expressions:
 - a. Evaluate whether the left side is a method call expression with a receiver's type that is a subtype of `java.util.Map`.
 - b. Evaluate whether the right side is a null expression.

JUST PLUG IN
Plug-ins are flexible, have a simple interface, can run at various points in the compilation pipeline, and are specific to javac.

We now need to create the class `CodePatternTaskListener`, which implements a `TaskListener`. A `TaskListener` has two methods, `started()` and `finished()`, which are called, respectively, before and after certain events. These events are encapsulated in a `TaskEvent`

Step 3: Create the AST visitor. Next, we need to write the logic to locate the code pattern and report it. How do we do that? Thankfully, a task event provides us with a [CompilationUnitTree](#), which represents the cur-

```
public interface Plugin {
    public String getName();
    public void call(JavacTask task, String[] pluginArgs);
}
```

 [Download all listings in this issue as text](#)

Our code will need to traverse this tree, locate a binary node, and evaluate the node's left and right children. This sounds like a visitor pattern job. The Compiler Tree API provides us with a ready-made visitor class designed for such tasks: `com.sun.source.util.TreeScanner<R, P>`. It visits all the nodes in the AST.

All we have left to do is to override `visitBinary(BinaryTreeNode, P p)` and write the logic to verify the code pattern. The full code of the visitor class with inlined comments is provided in **Listings 8a–8d**.

ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// JANUARY/FEBRUARY 2013

JAVA IN ACTION

JAVA TECH

ABOUT US


-

f


ava
net

- log





MAKE THE FUTURE JAVA





FIND YOUR JUG HERE

One of the most elevating things in the world is to build up a community where you can hang out with your geek friends, educate each other, create values, and give experience to you members.

Csaba Toth
Nashville, TN Java Users' Group (NJUG)

LEARN MORE







The New javax.cache Caching Standard

JAVA IN ACTION

JAVA TECH

ABOUT US



Java
net

bloo



58

While caching seems like a simple topic, **there are subtleties.**



Finally, the affordability of servers with memory capacities of 1 TB and higher, combined with vendor innovation to utilize that memory for cache storage, resulted in a new trend in which the cache has increased operational significance. Instead of caching just part of a data set, the entire data set is placed in cache and is used as an authoritative source of information—the cache, in essence, becomes the operational store for the application. In this use

Each of these areas has requirements that the standard must deal with.

At the time of this writing, JSR 107 is planned to be included in Java EE 7 and developed by [JSR 342](#). Java EE 7 is due to be finalized in 2013. In the meantime, [javax.cache](#) will work in Java SE 6 and higher and in Java EE 6 environments as well as with Spring and other popular environments.

The following vendors are either active members of the Expert Group or have expressed interest in implementing the specification:

- Terracotta (Ehcache)
- Oracle (Oracle Coherence)
- JBoss (Infinispan)
- IBM (ExtremeScale)
- SpringSource (Gemfire)
- GridGain
- TMax
- Fujitsu (Interstage XTP)

Terracotta, Oracle, and JBoss are all planning releases upon finalization of the spec. In addition, Spring plans to support JSR 107 closer to its final release.

From a design point of view, the basic concept is a **CacheManager** that holds and controls a collec-

tion of **Cache** instances that have entries. The API can be thought of as maplike with the following additional features:

- Atomic operations, similar to [java.util.ConcurrentMap](#)
- Read-through caching
- Write-through caching
- Cache event listeners
- Statistics
- Transactions including all isolation levels
- Caching annotations
- Generic caches that hold a defined key and value type
- Support for storage by reference (applicable to on-heap caches only) and storage by value

Rather than split the specification into a number of editions targeted at different user constituencies, such as Java SE or Spring and Java EE, we have taken a different approach.

First, for Java SE-style caching, there are no dependencies. And for Spring and Java EE, where you might want to use annotations and transactions, the dependencies will be satisfied by those frameworks.

Second, we have a capabilities API via `ServiceProvider.isSupported(OptionalFeature feature)` so you can determine at runtime the capabilities of the implementation. Optional features

LISTING 6

```
<dependency>
  <groupId>javax.cache</groupId>
  <artifactId>cache-api</artifactId>
  <version>0.5</version>
</dependency>
```



[Download all listings in this issue as text](#)

include the following:

- `storeByReference` (`storeByValue` is the default)
- Transactions
- Annotations

This makes it possible for an implementation to support the specification without necessarily supporting all the features, and it allows end users and frameworks to discover the features so they can dynamically configure appropriate usage.

While the specification does not mandate a particular distributed topology, it is cognizant of the fact that data storage might be distributed. We have one API that covers both usages but is sensitive to distributed concerns. We do not have high-network-cost maplike methods, such as `keySet()` and `values()`. And we generally prefer

zero or low-cost return types. So while `Map` has `V put(K key, V value)`, `java.cache.Cache` has `void put (K key, V value)`.

Caches contain data shared by multiple threads, which might themselves be running in different container applications or Open Services Gateway initiative (OSGi) bundles within one Java Virtual Machine (JVM) and might be distributed across multiple JVMs in a cluster. This makes class loading tricky.

We have addressed this problem. When a **CacheManager** is created, a class loader can be specified. If none is specified, the implementation provides a default. In either case, object deserialization will use the **CacheManager** instance's class loader.

This is a big improvement over the approach taken by caches

ORACLE®



Implement login authentication using declarative and programmatic security.

So far, the user is not going to be authenticated. Without authentication, the current user is not going to be identified as a **Principal** with its configured roles membership. In the case of GlassFish, a nonauthenticated user is associated with a **Principal** with the **ANONYMOUS** name. Fortunately, most of the Web frameworks, and so also the JAX-RS API, are Servlet-based. Servlets come with an easy way to authenticate a user with a single invocation of the method **HttpServletRequest.authenticate**. The entire procedure can be



PHOTOGRAPH BY
THOMAS EINBERGER/
GETTY IMAGES

extracted into a reusable Web filter, as shown in **Listing 3**.

With the annotation `@WebFilter("/")`, the filter intercepts all requests. The crux of the `Authenticator` filter is the invocation of the `authenticate` method.

Where Annotations End and XML Starts

In the old Java EE tradition, no single XML line has been written so far. Unfortunately, we will have to specify a few lines in `web.xml` (see **Listing 4**) to configure which authentication method will be used and from which realm the authentication and authorization information is fetched.

The `error-page` section is needed only to display for the user a nicer message than an “Internal Server Error 500” message. Because of its flexibility, form-based login configuration is the most popular. To pass the username and password to the system, you only have to use `j_security_check` as a name of the action and `j_username` and `j_password` to pass the username and password, respectively. In our example, the form is implemented as a static HTML page, as shown in **Listing 5**.

Of course, you could equally well pass the information directly with the `POST` call programmatically. You are not limited to using

a static HTML page to perform the authentication.

What Is file?

Within the `web.xml` deployment descriptor, a realm with the name `file` is referenced. A realm is a user repository containing the security credentials as well as the roles membership. A concrete realm realization is not specified by the Java EE specification. Usually, application servers tend to support file, database, and LDAP realms out of the box, and they support a large number of extensions to cover a wide variety of legacy resources. For demo purposes, we’ll later create a file realm with the name `file` and a user with the name `james` and membership to the `dukes` role.

As an enterprise developer, you cannot fully rely on concrete names existing in the production repository. Instead of hardcoding volatile role names into your code, you can use fictive role names and map them to the actual ones in the deployment descriptor.

Logical roles that come with the application are called *groups*. The Java EE spec foresees the mapping of roles to groups in the deployment descriptor. Of course, such a decoupling works only in cases where roles and groups with different names are semantically equivalent. If you can influence

LISTING 1

LISTING 2

LISTING 3

LISTING 4

LISTING 5

```
@Stateless
@Path("wisdom")
@Produces("text/plain") // MediaType.TEXT_PLAIN
public class WisdomResource {

    @Inject
    WisdomStorage storage;

    @GET
    public String wisdom() {
        return storage.wisdom();
    }

    @POST
    @Consumes(MediaType.APPLICATION_FORM_URLENCODED)
    @Produces(MediaType.TEXT_HTML)
    public String wisdom(
        @FormParam("wisdom") String wisdom) {
        return storage.wisdom(wisdom);
    }
}
```



[Download all listings in this issue as text](#)

the actual role naming in the target realm, you can also conventionally use the roles as groups and omit the mapping. In the case

of GlassFish, an automatic role-to-group mapping can be specified directly in the security realm configuration.



ity in our example, so we will use only generic permissions:

```
public enum Permission {
    READ,WRITE,EXECUTE;
}
```

In the real world, you can attach to a **Principal** whatever is beneficial for the realization of your requirements—it doesn't have to be a **Permission**. It can be any flat or hierarchical data structure. Usually the custom permissions are going to be maintained in a persistent store separately. For demonstration purposes, the **Permission** instances are maintained in an in-memory custom realm implemented as a singleton EJB bean, as shown in **Listing 8**.

In the real world,
you could extend the

InMemoryPermissionsRealm to

access the persistent store and cache the results. For reasons of convenience, the generic set of `Permissions` is wrapped with a more meaningful class representing the user, as shown in **Listing 9**.

A user populated with a custom set of permissions needs to be conveniently exposed to the application code in order to be useful. For the population of the user instance with the externally stored permissions, the name of the currently active user is needed. The class `UserProvider` uses the username stored in the injected `Principal` instance to instantiate a `User` with the `Permission` instances from the `InMemoryPermissionsRealm` (see **Listing 10**).

Now a user instance with custom permissions can be conveniently injected into any Java EE component:

```
public class SomeComponent {
    @Inject
    User currentUser;
}
```

Declarative Authorization with Even More Aspects

Although a **User** instance populated with custom permissions can be injected to all instances participating in the call stack, the handling is still insufficiently convenient. You would have to ask the **User** for authorization each time you are planning to invoke a guarded method. However, authentication and authorization are one of the few killer use cases

KILLER USE CASE

Authentication and authorization are one of the few killer use cases that **can be solved with aspects.**

LISTING 8 / LISTING 9 / LISTING 10 / LISTING 11 / LISTING 12

```
@Singleton
@ConcurrencyManagement(ConcurrencyManagementType.BEAN)
public class InMemoryPermissionsRealm {

    private Map<String, EnumSet<Permission>> customStore;

    @PostConstruct
    public void populateRealm(){
        this.customStore =
            new HashMap<String, EnumSet<Permission>>();
        this.customStore.put("james",
            EnumSet.allOf(Permission.class));
        this.customStore.put("blogger",
            EnumSet.noneOf(Permission.class));
    }

    public EnumSet<Permission> getPermissionForPrincipal(
        String userName){
        EnumSet<Permission> configuredPermissions =
            this.customStore.get(userName);
        if(configuredPermissions != null) {
            return configuredPermissions;
        }
        else {
            return EnumSet.noneOf(Permission.class);
        }
    }
}
```

 [Download all listings in this issue as text](#)

that can be solved with aspects. Instead of repeatedly asking the **User** for permission, a guarded method can be denoted with a plain annotation, as shown in **Listing 11**.

The `AllowedTo` annotation carries a `Permission` array and can be directly applied on business methods (see **Listing 12**). The actual checks were factored out into an interceptor, which compares the

JAVA IN ACTION

JAVA TECH

ABOUT US

0

f



ava
net

log

67

LEARN MORE

-
- MAKE THE
FUTURE
JAVA
-
- # FIND YOUR JUG HERE
- CEJUG is celebrating our 10-year anniversary in 2012! We follow Java technology with passion, share knowledge with pleasure, and create opportunities for students and professionals with ambition.
- Hildeberto Mendonça
- The Ceará Java User Group (CEJUG)
- LEARN MORE
- ORACLE®

integration with a social network is no different from integration with any other external Web service. Via a cURL command, the information can easily be retrieved using command-line requests, as shown in **Listing 1**.

The first call will retrieve tweets that are tagged with `java` and return the result in JSON format. The second call will return the public part of the profile of the Java Champions group on Facebook. A very basic Java program can be used to query Twitter and Facebook in order to retrieve information that is accessible without authentication, as shown in **Listing 2**.

Authentication

Typically, information that is available without a user having to log in on a Website is also available via an anonymous, nonauthenticated API call. Thus, this information can be retrieved easily by Java applications. It should be noted, though, that most social media sites impose volume restrictions. For example, you can make only 2,000 API calls per day or you can track only 10 users simultaneously.

Private—and often more interesting—information is accessible only by using authentication that involves three parties:

- The social network

- Your application
 - The end user
- Authenticated access to resources might seem more complex, but it comes with a number of benefits. The application knows the end user and has access to interesting information that can be valuable for its own purposes. To access user-specific information, the social network often requires that your application does this on behalf of the user. Typically, OAuth 1, OAuth 2, or a similar protocol is used for this. It is beyond the scope of this article to explain OAuth in depth. Interested readers can refer to the specification document for [OAuth 1](#) and to the draft version of OAuth 2.

In general, an application is allowed to make private calls only when the end user allowed it (see **Figure 1**). This explains messages such as “this application wants to post on your timeline” and “this application wants to read your e-mail.”

Rather than asking the user for permission for every single call, the application is often granted one user access token per user by a specific social network. All requests that the application makes on behalf of a specific user are then signed using that token. The social network then verifies whether the token provided by the

LISTING 1

LISTING 2

```
curl "http://api.twitter.com/1/users/lookup.json?screen_name=java"
curl "http://graph.facebook.com/javaChampions"
```



Download all listings in this issue as text

application grants sufficient access rights to the requested resource, for example, to read the social graph or to post tweets.

Obtaining a user access token is a multistep process, which is described here for OAuth 1.0. In

a browser application, users are redirected to a login page at the social network where they grant or deny permissions. The social network and the application communicate via a multiphase protocol, and the end result is the applica-

tion obtaining a user access token. Users are then redirected from the application to the social network and back. Schematically, this is shown in **Figure 2**.

Before an application can request that users authenticate themselves with social media, the application needs to be registered with the social network. Every social network has its own registration process, but the following are a number of common concepts:

- The application needs to provide a callback URL that will be called by the social network upon successful authentication.
- The application is granted a **consumerKey** and a **consumer Secret** or a public key and a private key; terminologies tend to vary between different social media. This key combination is used when signing requests.

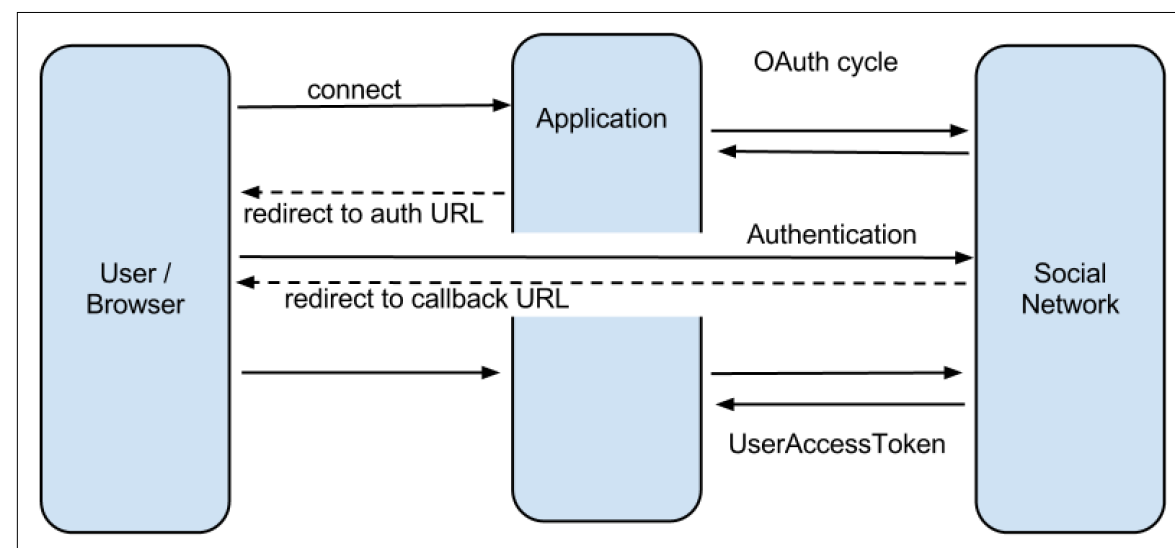


Figure 2

DaliCore

DaliCore is an open source project that adds the concepts of user, content, group, and permission on top of the Java EE 6 specification. The goal of DaliCore is to make it easier to develop applications that involve users and content, organize them in groups, and assign and check permissions. A high-level overview of the DaliCore architecture is shown in **Figure 3**.

DaliCore defines the concept of a user in the class `com.lodgon.dali.core.entity.User`. Operations on the user concept are defined in the stateless session bean `com.lodgon.dali.core.ejb.UserBean`.

The DaliCore project contains a module **DaliCoreSocial**, which is particularly useful in facilitating the above-mentioned process of obtaining user access tokens for

different social media. By using [DaliCoreSocial](#), an application developer is shielded from the rather complex flow that is required for authenticating a user with a social network.

A user of a social network corresponds to an instance of `com.lodgon.dali.core.social.entity.OnlineAccount`. A single user can be linked to zero or more `OnlineAccount` instances (for example, an `OnlineAccount` for Facebook and an `OnlineAccount` for Twitter).

The business functionality provided by **DaliCoreSocial** is available in the stateless session bean **com.lodgon.dali.core.social.ejb.SocialBean**. This session bean contains functionality for finding which **OnlineAccount** instances are coupled with a particular **User** instance, for example:

- findOnlineAccount
- findOnlineAccountByUser

Also, **DaliCoreSocial** provides functionality that applications can use to retrieve information from a social network or to push information to a social network, for example:

- getFriends
- getStatus

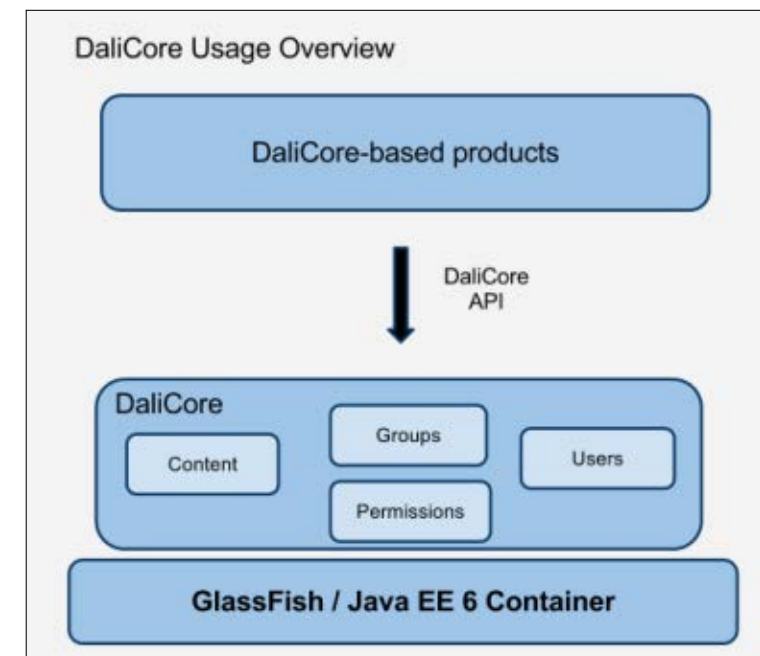


Figure 3

- `getStream`
- `postStatus`

In order to couple an **OnlineAccount** to a **User** or create a new **User** with an **OnlineAccount**, the end user needs to connect with a social network. As mentioned before, an application needs to register itself with each of the social media it wants to communicate with. The registration process is specific to each social network. For example, registering your application with Twitter requires you to log in at <http://dev.twitter.com> and register the application, as shown in **Figure 4**.

After registering successfully with a social network, the application is granted a key and a secret. **DaliCoreSocial** requires these keys

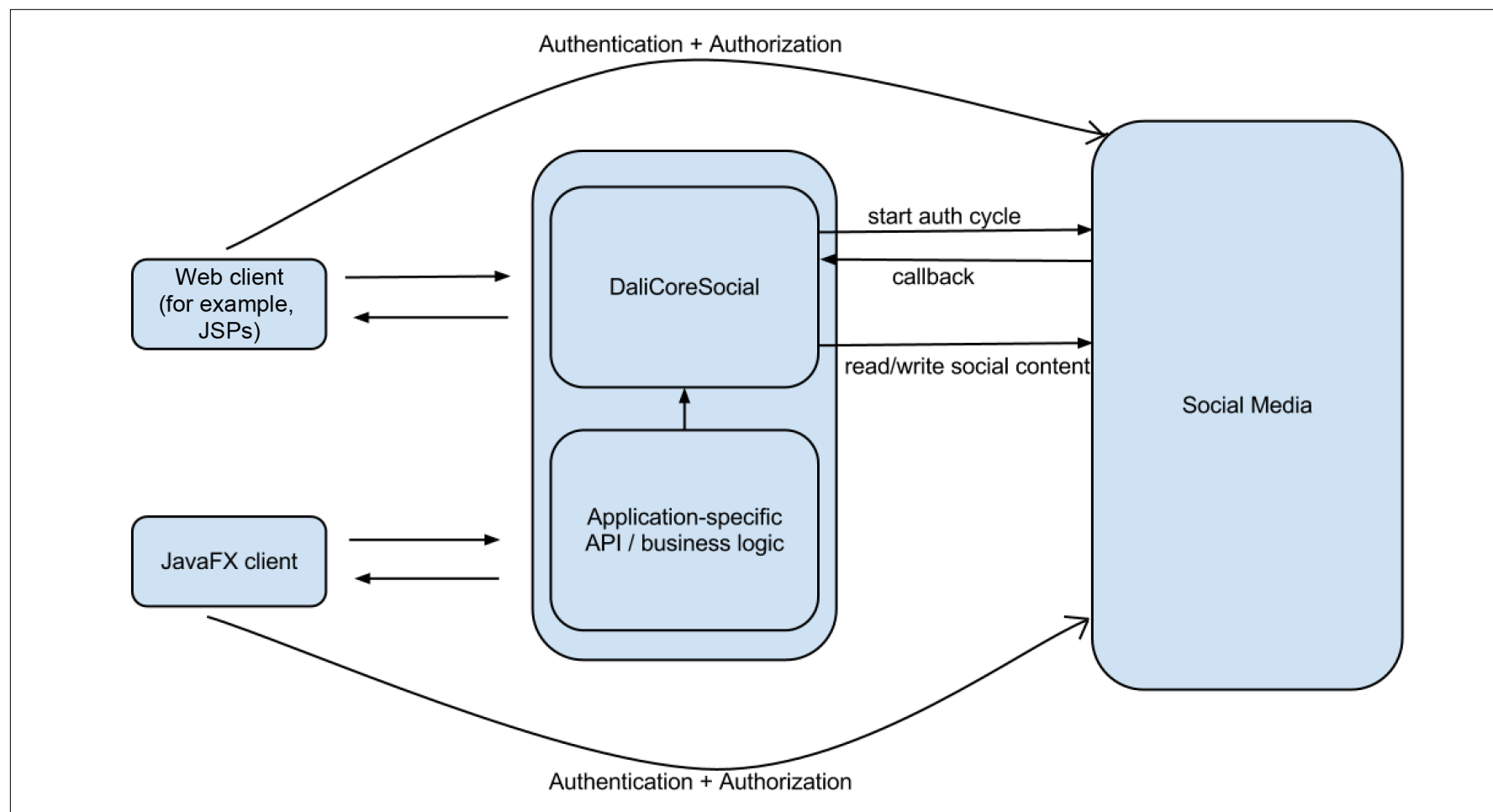


Figure 5

user corresponding to a specific user ID can be obtained by calling the following:

```
UserBean.getByUid(
    String uid);
```

The friends of a specific user can be retrieved by calling the following method, which will return the friends of the user for a specific network:

```
SocialBean.getFriends(  
    String uid, String network);
```

Our demo application contains two REST handlers:

- **Handler.java** is used by the Web example. The responses of the method calls are Jersey **Viewable** instances, which create HTML pages.
- **API.java** is used by the JavaFX example. The responses of the method calls are in XML or JSON format and can be processed by any client capable of doing XML/JSON processing.

The functionality provided by both REST handlers is similar.

Web Example

The Web application contains a home.jsp file that contains three images: a Facebook image, a Twitter image, and a Google+ image. Clicking one of those images will direct the user to the start of the login flow at one of the following, which will then direct the user to the specific social network login page:

- BASE/rest/dalicoresocial/connect/facebook?callback=BASE/rest/demo/callback
- BASE/rest/dalicoresocial/

```
connect/twitter?callback=BASE/
rest/demo/callback
```

- BASE/rest/dalicoresocial/connect/googleplus?callback=BASE/rest/demo/callback

The whole authentication flow (including retrieval of a user access token) is done by [DaliCoreSocial](#). When the user is authenticated and the social network allows our application to access the user information, an [OnlineAccount](#) is created and the user is redirected to [BASE/rest/demo/callback](#).

Our application contains a REST handler class named `Handler.java`, which is annotated with the `@Path` annotation to indicate that it listens to REST requests that start with "demo":

```
@Path("/demo")
@ManagedBean
public class Handler {
...
}
```

Note that the class is also annotated with `@ManagedBean` in order to easily use the stateless session beans `UserBean` and `SocialBean`:

```
@Inject
SocialBean socialBean;

@Inject
UserBean userBean;
```

The `callback` method is anno-

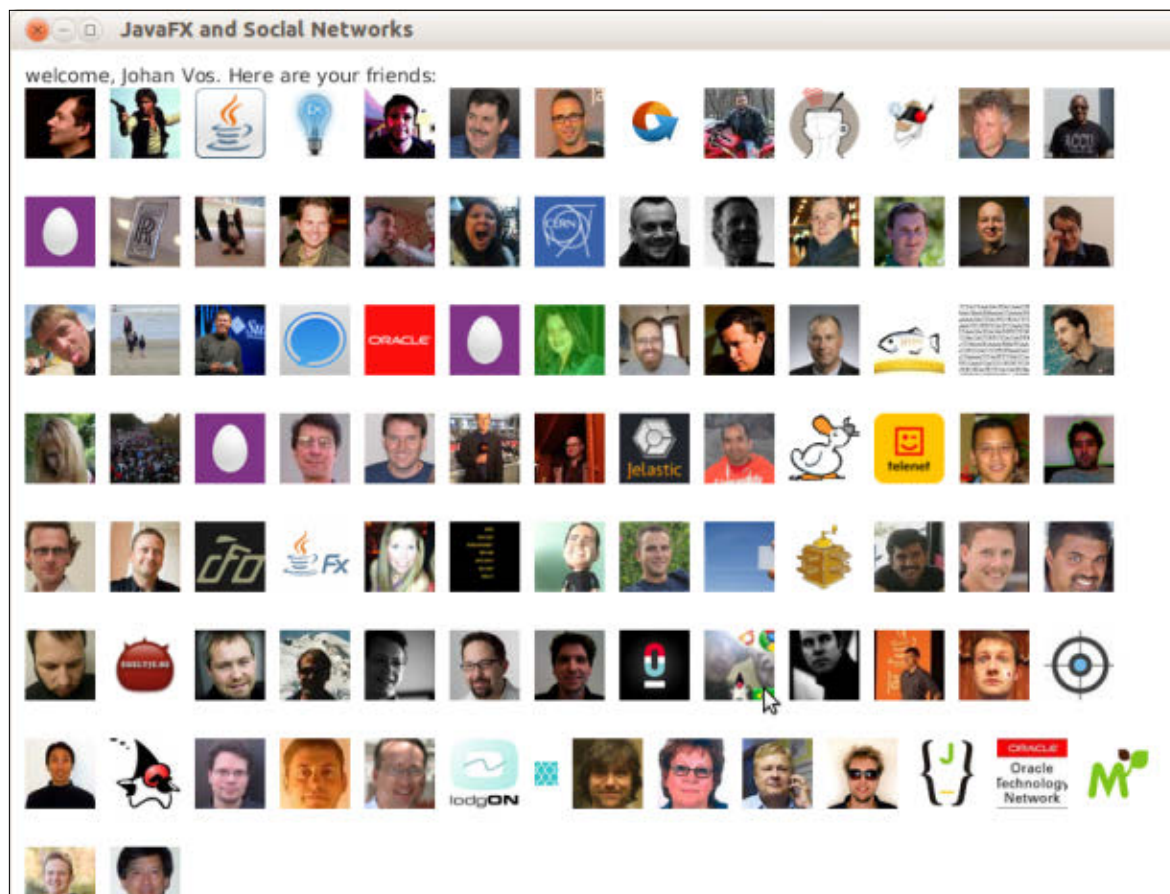


Figure 7

friends and create a [UserDetail](#) node for them. This node is added to a [FlowPane](#). The result of this application is shown in **Figure 7**.

Conclusion

In this article, we showed how your application can integrate with large, existing social network systems. Authentication, privacy, and permissions are important concepts for all social media. The [DaliCoreSocial](#) module deals with these concepts, and provides Java developers with an easy-to-use framework that can be used to

integrate social network users and functionality into your applications. It is important to note that the same approach works for Java EE applications (including Web clients) as well as for Java desktop applications or a combination of the two. [</article>](#)

LEARN MORE

- [Johan Vos' blog](#)
- [Johan Vos' YouTube screencast on DaliCoreSocial](#)

MAKE THE
FUTURE
JAVA



FIND YOUR JUG HERE

I have met so many amazing people through the London Java Community—friends, mentors, new colleagues—and through it I have achieved much more than I could have on my own.

Trisha Gee
London Java Community (LJC)

[LEARN MORE](#)



ORACLE®



VIKRAM GOYAL



IMS Services API—Getting Started with JSR 281

Learn how to bring IP Multimedia Subsystem services to Java-enabled devices.

The IP Multimedia Subsystem (IMS) involves the evolution of very basic telephone and data exchanges into a modern exchange system, powered by a plethora of high-capacity devices and networking lines. Driven by the recent explosion in internet and high-end devices, Java ME, which is installed on billions of devices worldwide, is strategically placed to take advantage of this.

Here, I offer a brief introduction to IMS technology, followed by a discussion of [JSR 281](#), the IMS Services API, which attempts to bring the IMS to Java-enabled devices. Finally, I will provide some simple code using the latest [Java ME SDK for mobile phones](#) (version 3.2) and the LG Session Initiation Protocol (SIP) server, which

provides an implementation of this API.

Let's get started.

What Is the IMS?

The IMS is an architectural framework that helps with delivering multimedia services that use the Internet Protocol (IP). It was originally defined by the 3GPP (3rd Generation Partnership Project) as a standardized way for wireless mobile devices to distribute internet-based services. However, it is broader now and covers multiple network and media types.

The architecture defines a centralized exchange-type of system that advertises its services. End users can connect their devices to this centralized service and gain individual identities. Multimedia-based messages and data are sent to these identities

via pipes that provide for simultaneous transfer of the data. Therefore, instead of an old-style telephone exchange system, which provides for only one kind of media transfer (voice), the IMS can easily handle multiple media transfers simultaneously.

This exchange of data can take place between services without a human element present, as in video or audio streaming where an end user consumes the data without needing to have a physical entity present at the other end. The end user is dialing into a service that would have advertised itself previously with its own unique identity. Business-to-business transactions are also possible, as long as individual identities have been established and a connection is possible based on

pre-existing criteria and contracts.

Identity on an IMS network is established using [SIP](#) (RFC 3261—essentially, your e-mail address) or a “tel” URI (RFC 3966—essentially, your phone number). An end user or service may have multiple such identities, and the user can be reached on one, some, or all of these identities, *at the same time*. This is the beauty of an IMS-based system.

I have used the example of telephony so far, which would be the most common use. However, the structure of IMS precludes no particular data and is, therefore, media agnostic. You could easily use movies delivered over this architecture to illustrate its usage.

A commercial IMS system would control the functionality of the whole network

PHOTOGRAPH BY
JONATHAN WOOD/
GETTY IMAGES

and provide the basic structure to get the system going. Thus, it would seamlessly integrate all available networks and provide authentication and authorization. Further, it could actually look up new networks and services and provide them to the end user for use. A recent example of this service is the Push-To-Talk feature that has been tried successfully by several telecommunications companies. Each IMS system should be able to advertise its services (or capabilities) when new subscribers connect to it.

If an IMS device provides new capabilities, these need to be separately identified. For each service that is standard, the 3GPP defines a standardized identifier called the IMS Communication Service Identifier (ICSI). For services that are new or localized to your application, the IMS Application Reference Identifier (IARI) is recommended.

One final point to note about the IMS is that it doesn't matter how a user connects to an IMS network; it always provides the same services to the end user. So a user could connect to a local IMS network via Wi-Fi at home, to a 3G or 4G network while traveling, and to a fixed-line network when at the office. At each place, the user experience within the IMS will be the same.

Java ME and the IMS

Java is present on a plethora of devices (not just mobile phones, but set-top boxes as well), so it is ideally suited to use the IMS. JSR 281, a step in this direction, defines an API that can be used to create an IMS-ready application that can sit on any Java-enabled device that implements this API.

A device that implements JSR 281 provides the *IMS engine*—an execution environment that an IMS-ready application can interface with. The application itself is defined with the following parameters:

- IMS application identifier (AppId), which is a string that identifies your application (potentially from among other IMS applications on the same device)
 - The MIDlet, which handles the application logic and—in JSR 281 lingo—is said to be the “owning Java application”
- The IMS engine stores these properties in the registry within the IMS device. This registry is used to identify the properties that define the capabilities of the IMS application.
- Application developers need to

define this registry by writing the capabilities within the Java Application Descriptor (JAD) or Java archive (JAR) manifest file. This is the static method of installing an IMS application on an IMS-capable device. JSR 281 also allows developers to dynamically register their application from within the MIDlet by using the classes in the [javax.microedition.ims](#) package.

The capabilities are defined in the registry by using at least one of the properties described in **Table 1: StreamMedia, BasicMedia, FramedMedia, Event, or CoreService**. The capability declara-

PROPERTY NAME	PROPERTY MEANING	EXAMPLE PROPERTY VALUES	SAMPLE DECLARATION
StreamMedia	MEANS THAT YOUR APPLICATION CAN STREAM AUDIO, VIDEO, OR BOTH.	Audio OR Video OR Audio Video	MicroEdition-IMS-1-Stream: Audio Video
BasicMedia	MEANS THAT YOUR APPLICATION CAN TRANSFER OR HANDLE MEDIA OF THE TYPES DEFINED BY THIS PROPERTY VALUE.	image/png, text/plain, application/myApp	MicroEdition-IMS-1-Basic: image/png
FramedMedia	SPECIFIES THAT THE APPLICATION CAN TRANSFER APPLICATION DATA-LIKE MESSAGES AND FILES. YOU NEED TO DEFINE THE CONTENT TYPES AND THE SIZE.	text/plain image/png, 4096	MicroEdition-IMS-1-Framed: text/plain image/png, 4096
Event	SPECIFIES THAT YOUR APPLICATION USES THE PUBLICATION AND SUBSCRIPTION CLASSES TO DECLARE EVENTS RELEVANT TO AN IMS APPLICATION.	presence	MicroEdition-IMS-1-Event
CoreService	DECLARES THE CORE SERVICES THAT THE APPLICATION SUPPORTS AND DEFINES ANY COMPOSITE PROPERTIES AND SERVICES THAT ARE SUPPORTED.	urn:URN-3gpp.org.3gpp.icsi;require;explicit FOR ICSI AND urn:IMSAPI.com.myCompany.iari.myApp FOR IARI	MicroEdition-IMS-1-CoreService-1

Table 1

Use the Lightweight UI Toolkit on Nokia Series 40 phones.

BIO

With an estimated 675 million Series 40 phones in active use every day, there is a huge potential market for your apps. With support for in-app purchase and in-app

By providing a set of rich UI components, LWUIT

Nokia has a long tradition of supporting Java in its platforms and creating robust



LISTING 1

 [Download all listings in this issue as text](#)

lower right of the screen and in the Options menu, but the Options menu is repositioned in the full-touch UI. The primary action (called the *default command* in

ORACLE.COM/JAVAMAGAZINE ////////////////////////////////// JANUARY/FEBRUARY 2013



JAVA TECH

ABOUT US

This is obviously a contrived example, but it does create for us a failure scenario—one that allows for at least a certain amount of



strophic failure and we need to create a new one to do message processing).

- Stop this actor.
- Escalate the issue to this actor's own supervisor (which, in the case of `HelloActor`, will be the Akka system runtime) by rethrowing the exception to it. For example, by default, a


In cases where we want entirely new behavior, we need to override the `supervisorStrategy` value. The canonical way to do this is within the actor subtype itself, as shown in **Listing 4**.

There's more to the `SupervisorStrategy` constructor, but the parameters `maxNrOfRetries` and `withinTimeRange` are fairly self-explanatory, are described in the Scaladocs for Akka, and have some generally useful defaults.

Notice that the code in **Listing 4** uses the *context value*, which is a bound value built in to every actor that provides a reference to the actor system in which it executes. This is a reference to an **ActorContext** object that has

- Resume the failed actor's message processing (meaning the actor didn't suffer a catastrophic failure and we can keep using it).
- Restart the failed actor (meaning the actor suffered a cata-

LISTING 4

 [Download all listings in this issue as text](#)

- **actorOf**: This method is used in **Listing 4** to create another actor that will be a child of this actor (the one doing the **actorOf** call).
- **children**: This returns an **Iterable** of references to all the child

- **parent**: This is the parent of this actor.
- **self**: This is a reference to the actor itself—this is a little different from the **this** reference, because the **self** reference is to the **ActorRef** that wraps the actor, rather than to the raw

method, on the other hand, is less syntactically short but takes the timeout parameter explicitly, which is sometimes a little easier to read.)

Once that's done, the returned object from the `? call` is mapped to a particular future-result type. (In this case, because we're expecting a string back, it needs to be mapped back to a `Future[String]` type.) Then, finally, as shown in **Listing 9**, we need to establish how long we're willing to wait for that `Future` object to be populated with the result before the Akka API will allow us to actually harvest that result.

Why all the fuss? Michael Nygard, in his book *Release It!*, describes in great detail why unbounded waits are a bane to production-quality code. So I won't get into the details here, but in summation, an unbounded wait is essentially a deadlock or a production "hang" waiting to happen. The repeated use of timeout values here tells Akka exactly how long to wait for each potentially hanging operation before recovering and continuing execution (giving you the chance to recover from the lost message or response).

Note, though, that certain import statements (see **Listing 10**) must appear prior to the code in **Listing 9** for the code to compile, because neither the `ask` method

nor the `? method` appears on the `Actor` API. Failing to use these import statements will result in some truly misleading compiler messages.

Needless to say, Akka is making it a lot easier to use one-way messages, and despite the additional work, building redundant and fault-tolerant systems fares better when doing exactly that.

Get This to Phil in Accounting!

Seeing these messages flying from one actor to another might lead some readers to wonder if this isn't a replacement for Java Message Service (JMS) or some other kind of messaging-oriented middleware system. On the surface, yes, there appear to be some similarities, particularly when routing messages (our next topic), but similarities end there on the surface, really. A messaging-oriented middleware system is generally about making the messages and their delivery a top-level concern as something the developer wants to see and deal with, usually for reasons of durability, persistence, and so on. In an actor-based system, the messages are almost an implementation detail; they are used solely to help break the blocking nature of a method call and (in the case of the mailboxes themselves) are not

LISTING 9 LISTING 10 LISTING 11

```
val ma = system.actorOf(Props[MockingActor])
implicit val timeout = Timeout(5 seconds)
val future = (ma ? "Hello").mapTo[String]
val result = Await.result(future, 5 seconds)
println("Received result: " + result)
```



[Download all listings in this issue as text](#)

something the developer wants or needs to deal with.

However, there are interesting parallels between the two kinds of systems. One parallel is that of creating a "workflow" or "pipeline" of actors that can work together to accomplish a common goal. It might sound "retro," but building large systems out of small pieces—for example, by using UNIX command-line tools such as `grep`, `sed`, `awk`, and `more`—is by far a more resilient and robust way to achieve long-term success than by building monolithic large "chunks" of code.

In some cases, these chunks of code will be distinct and separate actor types (a `PrintlnActor`, a `FibonacciActor`, a `FactorialActor`, or—if we want to get out of the mathematics domain—a `ProcessRequestActor`, a `TransferMoneyActor`, a

`NotifyAccountActor`, and so on).

And for some of these, linear sequences of steps through each (for example, A to B to C to D) will work, in which case, we can simply pass the "next" actor in to the previous one. But for others, some of the steps will be processed in parallel or in some combination or hybrid thereof. This entire subject is called *routing* in Akka, and while there's no way we can exhaustively cover all the different possibilities, we can explore a few simple scenarios.

Consider creating an actor specifically for printing to the console, as shown in **Listing 11**. In this particular example, we want to have five different instances of the `PrintlnActor` (which we'll be able to verify by seeing the "created!" message printed when each one is instantiated). More importantly, when we send 10 messages, we


```
val pp = system.actorOf(
  Props[PingPongActor])
pp ! Start
pp ! Start
pp ! Start
```

It begins!
PING
PONG


We Need to Shoot a Scene in a Place Far, Far Away

for this is simple: The programmer's approach to remote actors is, essentially, exactly the same as that for local actors, which is simply sending messages. The only things that are different are establishing a configuration that allows for remote communication (which consists of the right entries in a custom text file format that Akka can find when running your code) and specifying how to find (or create) the actor on the remote system. Neither of these tasks substantively changes the nature of your interaction with the actor nor the design of the actor-based system as a whole.

In order to keep the external requirements to a minimum, I'm going to embed the configuration in the code and read it directly using a **ConfigFactory** (which, by the way, is not technically part of Scala; the Scaladoc is on the Typesafe Website). This approach is obviously not recommended

LISTING 17

```
akka {  
  actor {  
    provider = "akka.remote.RemoteActorRefProvider"  
  }  
  remote {  
    transport = "akka.remote.netty.NettyRemoteTransport"  
    netty {  
      hostname = "127.0.0.1"  
      port = 2552  
    }  
  }  
}
```

 [Download all listings in this issue as text](#)

Next, we need to establish the server on the remote machine, as shown in **Listing 17**. (Again, this will be the same machine we're running on to keep things simple.)

Aside from the configuration passed in to the `ActorSystem`, the server code is fairly standard Akka code. It creates the actor using `actorOf`, except this time, we give it a name in order to be able to find this actor from the client. We then block on an incoming keystroke (using `System.in`) just to keep the process up and running. (Again, this is not really recommended for production code.)

The client, then, is also almost just as deceptively simple (see **Listing 18**). The only thing that's new is that the method `actorFor` is used instead of `actorOf`, and the parameter to it is a URI describing where to find the remote actor. The format of the URI is fairly self-describing, as most URIs are, and most of the information was contained in the server's configuration:

- The scheme (**akka**)
- The server's **ActorSystem** name (**server**)
- The server's host name (in this case, the raw IP address 127.0.0.1)
- The server's port (2552)
- The actor's path (**/user/actorName**, where **actorName** was the string passed in when the actor was created using **actorOf** on the server)

When building the code, note that the system will now depend

on a few more .jar files, notably the akka-remote JAR and the external dependencies netty-3.5.3.Final.jar and protobuf-java-2.4.1.jar (from the akka-2.0.3 release), all of which are already found in the Akka distribution but will likely need to be referenced somewhere in your build path or the project settings of your integrated development environment.

Conclusion

Akka is not, by any stretch of the imagination, a trivial system to master. The concepts are straightforward—message passing in a shared-nothing environment—but making the transition to building actor-oriented systems can be tricky. Just keep an eye on sending one-way messages, avoid blocking, and remember to let it fail! **</article>**

LEARN MORE

- [Akka Website](#)
- ["Java Champion Jonas Bonér Explains the Akka Platform"](#)

LISTING 18

```
object Client {
  def main(args: Array[String]): Unit = {
    val config = """
akka {
  actor {
    provider = "akka.remote.RemoteActorRefProvider"
  }
  remote {
    transport = "akka.remote.netty.NettyRemoteTransport"
    netty {
      hostname = "127.0.0.1"
      port = 2553
    }
  }
}
"""

    val system = ActorSystem("client",
      ConfigFactory.parseString(config))

    val actor =
      system.actorFor(
        "akka://server@127.0.0.1:2552/user/actorName")
    println("We have an actor: " + actor)
    actor ! "HELLO!!"
    println("Message sent")
  }
}
```




In the November/December 2012 issue, Jason Hunter and Boris Shukhat gave us a code challenge around porting JDBC code to use the Java Persistence API (JPA). They gave us code from a program that was redesigned to use JPA.

The correct answer is # 4. JPA commit() does a database rollback itself if it fails and then it closes the transaction. The correct code looks like this:

```
entityTransaction.begin();
try {entityManager.createNativeQuery(
sqlInsertStatement).executeUpdate();
    entityTransaction.commit();}
catch(RollbackException e) {log.info(
"Transaction failed and was rolled back");}
catch(RuntimeException e) {
if (entityTransaction.isActive()) {entityTransaction
.rollback();log.error(
"Transaction ended abnormally and was rolled back");}
}
```

This issue's code challenge comes from Simon Ritter, a Java evangelist at Oracle, who presents us with an embedded problem.

1 THE PROBLEM

Both the Connected Device Configuration (CDC) and the Connected Limited Device Configuration (CLDC) use the Connector framework to support generic input and output through a minimal-footprint API. CDC forms the basis of

Oracle Java ME Embedded Client, and CLDC forms the basis of Oracle Java ME Embedded.

In the following example, we use a DatagramConnection to set up a server that listens and processes messages from clients. The messages are of variable length, but never more than 256 bytes. When the application runs, some messages require multiple reads to get the whole message.

2 THE CODE

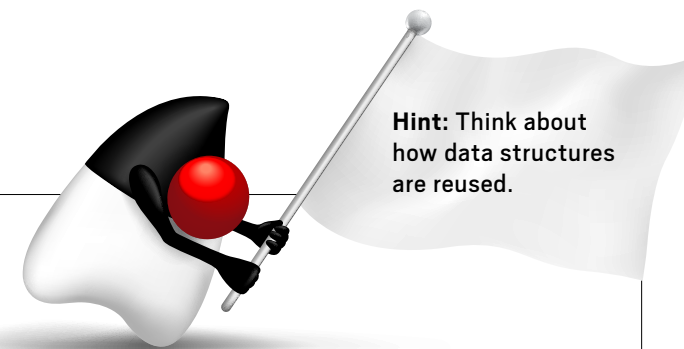
How should this code be modified so all messages are read in a single getData() call?

```
datagram = connection.newDatagram(256);

while (notDone) {
    connection.receive(datagram);
    data = datagram.getData();
    bytesReceived = datagram.getLength();
    // process datagram ...
}
```

3 WHAT'S THE FIX?

- 1) Ensure that the type of the connection is UDPDatagramConnection.
- 2) Move the call to connection.newDatagram inside the while loop.
- 3) Add a call to datagram.reset() before the call to connection.receive().
- 4) Add a call to datagram.setLength(256) before the call to connection.receive().



Hint: Think about how data structures are reused.

→ GOT THE ANSWER? Look for the answer in the next issue. Or submit your own code challenge! ←